AD-A255 441

PAGE

Form Approved
OMB No. 0704-0188

Public report... gathering and... collection of... Paperwork... per response, including the time for reviewing instructions, searching existing data sources on of information. Send comments regarding this burden estimate or any other aspect of this on Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson ...t and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		FINAL 1 Aug 88 - 31 Dec 91

4. TITLE AND SUBTITLE
"RECURSIVELY GENERATED NETWORKS & DYNAMICAL LEARNING (U)

5. FUNDING NUMBERS
61102F
2305/B3

(2)

6. AUTHOR(S)
Dr Eric Mjolsness

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Dept of Computer Science
Yale University
New Haven CT 06511

8. PERFORMING ORGANIZATION REPORT NUMBER
AFOSR-TR-

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM
Bldg 410
Bolling AFB DC 20332-6448

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFOSR-88-0240

DTIC
ELECTE
SEP 14 1992
S A D

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release;
Distribution unlimited

12b. DISTRIBUTION CODE
UL

13. ABSTRACT *(Maximum 200 words)*

Much of the research has been based on the premise is that mathematical methods and notation associated with constrained optimization should be used to specify a neural net, which can then be compiled to diverse implementations. But where do they get such a compiler? And what are the details of this mathematical notation? They have made substantial progress on these research questions: (1) They have developed methematical methods that can transform one algebraic NN description into another, more implementable one. These developments were attained by serious work in the applied mathematics of neural nets. They can form the basis of a neural compiler because they address most of the major NN compilation and implementation issues. But they do not yet suffice. (2) They have been accumulating the research in a neural simulator. It can be expanded into a semi-automatic compiler: a neural net design and implementation environment based on methematical methods. (3) They have developed a mathematical notation (not yet a formal language) for describing complex problem domains in terms of constrained optimization problems. The optimization problems can be solved by neural nets.

92-25081

14. SUBJECT TERMS

15. NUMBER OF PAGES
42

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

Final Technical Report
for AFOSR, 8/1/88-12/31/91

Eric Mjolsness
Department of Computer Science, Yale University

Abstract

We report on three years' research into artificial neural networks supported by the Air Force Office of Scientific Research under AFOSR grant 88-0240 from 1988 to 1991.

Contents

1 Introduction

In this final report, we will discuss three years' research supported by the Air Force Office of Scientific Research under AFOSR grant 88-0240 from 1988 to 1991. The discussion will be from the instrumental point of view of current and future research, represented by the successful renewal proposal [1] (of which this report is a summary), rather than from a historical point of view.

Much of our research has been based on the premise is that mathematical methods and notation associated with constrained optimization should be used to specify a neural net, which can then be compiled to diverse implementations. But where do we get such a compiler? And what are the details of this mathematical notation? We have made substantial progress on these research questions:

1. We have developed mathematical methods that can *transform* one algebraic NN description into another, more implementable one. These developments were attained by serious work in the applied mathematics of neural nets. They can form the basis of a neural compiler because they address most of the major NN compilation and implementation issues. But they do not yet suffice.

2. We have been accumulating the research in a neural simulator. It can be expanded into a semi-automatic compiler: a neural net design and implementation environment based on mathematical methods.

3. We have developed a mathematical notation (not yet a formal language) for describing complex problem domains in terms of constrained optimization problems. The optimization problems can be solved by neural nets as described in point 1. Our notation involves objective functions, L-system grammars, and maps between such objects.

The design and implementaion method outlined in these assertions is illustrated in Figure 1. We will outline our achievements in each of these areas of research, and their relationship to a larger plan of future work on the use of symbolic algebra in implementing neural networks, in the next section. The final section and the appendices will provide an entrance to more detailed expositions of the work.

2 Summary of Research Progress

2.1 Algebraic Transformations

Assertion 1. We have developed mathematical methods that can transform one algebraic NN description into another, more implementable one. These developments were attained by serious work in the applied mathematics of neural nets. They could form the basis of a neural compiler because they address most of the major NN compilation and implementation issues. But they do not yet suffice.

In [2] (included as an Appendix) we first introduced a catalog of fixedpoint-preserving transformations that could be applied to neural net objective functions, so as to reduce their cost or increase their implementability in some technology. The catalog is shown here as equation 1. This catalog is intended ultimately to provide the same kind of cookbook approach to neural net design that a table of integrals

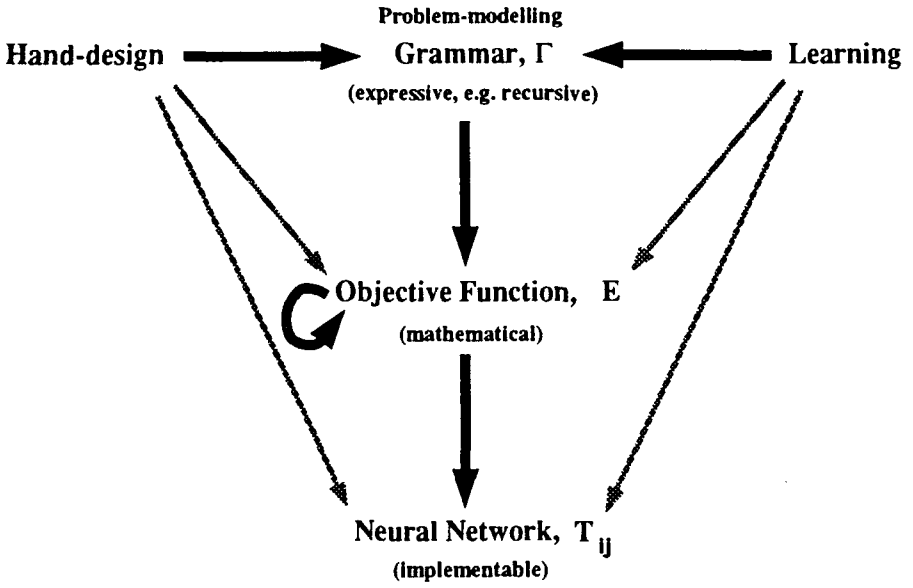


Figure 1: A neural network design methodology. Solid arrows constitute the recommended procedure. The arrow from Γ to E may be realized by approximations from statistical physics, such as Mean Field Theory. The circular arrow represents fixed-point preserving transformations of objective functions.

provides to integration: the intellectual work is not magically eliminated, but it is considerably reduced through the distilled widsom of generations of previous researchers. The simplest example discussed in [2] (using Rule 1.1) explains the then-standard trick for reducing a winner-take-all network from $O(N^2)$ connections to $O(N)$; that method has been used in a variety of analog neural networks chips including a stereopsis chip by Delbruck and Mead. This and several other cataloged algebraic transformation patterns could be shown to be Legendre transformations; others weren't of this class, but still preseved fixed points. All could be applied mechanically once the decision to use them was taken.

The limitations of the first catalog of transformations were clear: they applied to optimization-based ("Hopfield") nets and couldn't smooth out rough objective functions. So learning, convergence speed and global optimization were all questionable. Nevertheless they gave the neural net designer some remarkable capabilities.

Technology-Specific Tricks. For example special neural transfer functions, such as log and exponential, are available in CMOS [3]. Transformations could be used to shift the corresponding nonlinearity from a single variable to an entire expression in an objective function, greatly expanding the range of implementable objectives. Also the wiring complexity of circuits other than the winner-take-all net could be greatly reduced by introducing (semi-automatically) new linear interneurons that compute reusable expressions. Graph-matching networks were a subtle example, potentially central to high-level vision.

1.1	$\frac{1}{2}X^2$	$\rightarrow$	$X\sigma - \frac{1}{2}\sigma^2$
1.2	XY	$\rightarrow$	$X(\sigma-\tau)+Y(\sigma-\omega)-\frac{1}{2}\sigma^2+\frac{1}{2}\tau^2+\frac{1}{2}\omega^2$
1.3	XY	$\rightarrow$	$\frac{1}{2}X(\sigma-\tau)+\frac{1}{2}Y(\sigma+\tau)-\frac{1}{4}\sigma^2+\frac{1}{4}\tau^2$
2.1	$\int^X f(u)du$	$\rightarrow$	$X\sigma - \int^\sigma f^{-1}(u)du$ (f an invertable function)
2.2	e^X	$\rightarrow$	$(X+1)\sigma - \sigma\log\sigma$
2.3	$\lvert X\rvert\log\lvert X\rvert$	$\rightarrow$	$\lvert X\rvert(\sigma+1)-e^\sigma$
2.4	$\log\lvert X\rvert$	$\rightarrow$	$X\sigma - \log\lvert\sigma\rvert$
2.5	$\frac{1}{1+p}\lvert X\rvert^{1+p}$	$\rightarrow$	$X\sigma - \frac{1}{1+1/p}\lvert\sigma\rvert^{1+1/p}$ ($p\neq -1,0$).
3.1	$\int^X du\left[\int^Y dvg(u,v)\right]^{-1}(u)$	$\rightarrow$	$X\sigma - Y\tau + \int^\tau dv\left[\int^\sigma dug(u,v)\right]^{-1}(v)$ (If function inverses exist.)
3.2	$YF(X)$	$\rightarrow$	$-X\sigma + Y\tau + \sigma F^{-1}(\tau)$ (If $((F')^{-1})'$ exists.)
3.3	$Y\int^X f(u)du$	$\rightarrow$	$XY\sigma - Y\int^\sigma duf^{-1}(u)$ (If $(f^{-1})'$ exists.)
3.4	X/Y	$\rightarrow$	$X\sigma - Y\tau + \tau/\sigma$
4.1	$\min_\alpha X_\alpha$	$\rightarrow$	$\sum_\alpha X_\alpha\sigma_\alpha + C(\sum_\alpha\sigma_\alpha - 1)\lambda - \frac{C}{2}\lambda^2 + \sum_\alpha\phi(\sigma_\alpha)$ (Large C, and high-gain hysteresis-free barrier function ϕ which confines σ_α to $(0,1)$.)
4.2	$\max_\alpha X_\alpha$	$\rightarrow$	$\sum_\alpha X_\alpha\sigma_\alpha - C(\sum_\alpha\sigma_\alpha - 1)\lambda + \frac{C}{2}\lambda^2 - \sum_\alpha\phi(\sigma_\alpha)$ (Same conditions.)
4.3	$\prod_\alpha\lvert X_\alpha\rvert$	$\rightarrow$	$\sum_\alpha(\sigma\tau_\alpha - \lvert X_\alpha\rvert\omega_\alpha + \omega_\alpha e^{\tau_\alpha}) + \sigma(1-\log\sigma)$.
5.1	$\mathrm{tr}\ XY^T$	$\rightarrow$	$\mathrm{tr}\ X(\sigma-\tau)^T + \mathrm{tr}\ Y(\sigma-\omega)^T - \frac{1}{2}\mathrm{tr}\ \sigma\sigma^T + \frac{1}{2}\mathrm{tr}\ \tau\tau^T + \frac{1}{2}\mathrm{tr}\ \omega\omega^T$ (All matrices, possibly sparse.)
6.1	$E[\mathbf{v}]$	$\rightarrow$	$L[\dot{\mathbf{v}}\lvert\mathbf{q}] = \int dt\left(K[\dot{\mathbf{v}}\lvert\mathbf{v},\mathbf{q}] + \frac{dE}{dt}\right)$, $\delta L/\delta\dot{\mathbf{v}}(t) = 0$ ($\partial K/\partial\dot{\mathbf{v}} = 0 \Leftrightarrow \dot{\mathbf{v}} = 0$)
6.2	$E[v]$	$\rightarrow$	$\hat{E}[q] = \sum_i(\partial E/\partial v_i)\dot{v}_i + \hat{E}_{\mathrm{cost}}[q]$

(1)

Table of Algebraic Transformations. These transformations preserve fixed points of objectives, or summands thereof. X and Y are any algebraic expressions, containing any number of variables.

Parallel Computer Implementation. Furthermore, linear interneurons could be introduced between two previously connected neurons x and y:

$$E_{xy} = xy \quad \rightarrow \quad \hat{E}_{xy} = x(\sigma - \tau) + y(\sigma - \omega) - \frac{1}{2}\sigma^2 + \frac{1}{2}\tau^2 + \frac{1}{2}\omega^2. \tag{2}$$

This innocuous-looking transformation has great consequences, for the linear interneuron σ can be interpreted as the wire connecting up two different modules in a partitioned hardware implementation (e.g. two chips or two CPU's in a parallel computer). This interneuron can relax according an entirely different speed schedule from the other neurons, allowing a *communication channel* to be modelled. In fact the physical wire carrying σ may even be *multiplexed* with other connections between the same two modules. We have performed preliminary experiments [4] in which this transformation was used to split up a large neural net across 2 to 8 processors of an Encore Multimax running Linda, in which the observed substantial speedup resulted solely from our ability to vary the relative speed of the σ interneurons. In other words, this transformation allowed a neural net parallelization on a coarse-grained MIMD machine which would otherwise not have worked. (We will propose to continue this investigation.)

Controllable Dynamics. Also in the first catalog was a highly nontrivial transformation that replaces the objective function itself with a Lagrangian that governs the entire state space trajectory, including the dynamics, of a neural net, while still forcing convergence to locally optimal states according to the original objective. The central problem was to preserve the Lagrangian formalism as much as possible while allowing convergence to a fixed point; we did this by replacing the conventional functional derivative with a variant called the "greedy functional derivative".

Virtual Neurons and Attention. The aforementioned algebraic transformation from objective to Lagrangian can be used to model *virtual neurons and connections* and to optimize the kind of hardware *multiplexing* done in the Bell Labs ANNA chip, which is inevitable for any implementation in which a large network is mapped to a smaller but flexible circuit by using physical neurons to simulate different virtual neurons at different times. More generally the Lagrangian approach provides a "computational attention mechanism" for optimally choosing the most important part of an optimization problem to work on next. In [5] we applied this method to derive several kinds of attention windows for a two-dimensional surface reconstruction network, including sliding, jumping and rolling windows of attention. A combination of rolling and jumping is expected to be the most effective here; the investigation continues.

Multiscale Acceleration. Since the first catalog of transformations was published we have developed others. The question of *convergence speed* was addressed first, with the publication [6] of a multiscale acceleration technique for optimization neural nets. It is a generalization of the standard highly effective multigrid algorithm for solving partial differential equations, systems of ODEs, or just systems of linear equations. It requires very few assumptions on the neural network to be applied, and it may be viewed as a transformation which turns one fine-scale objective function into a set of compatible neural net objective functions at different scales, including the original one at the finest scale.

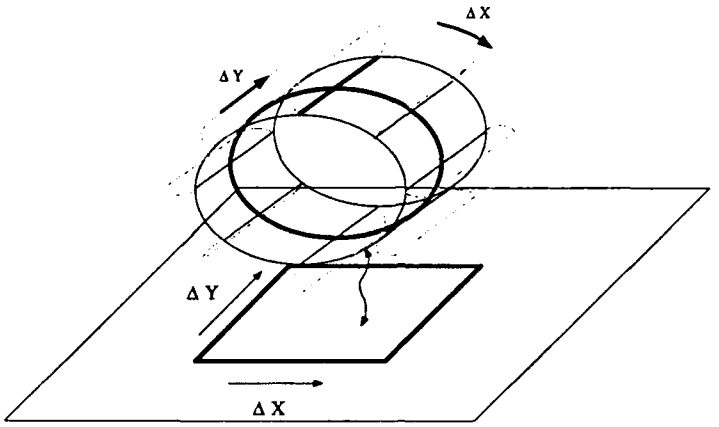


Figure 2: A rolling window of attention.

Communication Cost. Communication cost has been addressed by the development of a special-purpose message-routing network for Content Addressable Memories and other models of recognition in which the closest memory according to some metric is to be retrieved. Once again it may be viewed as an algebraic transformation, though on a very special form of objective function. This algorithm was put in its current form by Professor Bhatt at Yale, an expert on communication in parallel computers, and we are currently implementing it on a Connection Machine. The empirical timing results are consistent with the theoretical performance: simultaneously matching N inputs to N memories takes $O(N \log N)$ wires and $O(\log^2 N)$ time steps of a communication-bound feed-forward algorithm. We propose to continue this work. More generally, message routing and other standard communication problems arise for any sufficiently large or complex neural network implementation, but in an especially advantageous form since problem-specific knowledge may be used to improve low-level communication performance. There is no guarantee that biological networks use a general-purpose message router even if for reasons of algorithmic efficiency they must perform analogous functions.

Learning. Learning and especially learning in feed-forward nets has received remarkably short shrift in our research so far, since it was not on the main line of technical development. However we have known for two years that an algorithm extremely close to Pineda's recurrent backpropagation can be derived by transforming the standard squared-error objective function for learning. The difference is only in the dynamics; the set of fixed points is the same as Pineda's. Also feed-forward networks have a trivial objective function themselves [7], and hence can be handled this way. Since nobody needs another derivation of backpropagation, this method is of interest only insofar as it can be automated to

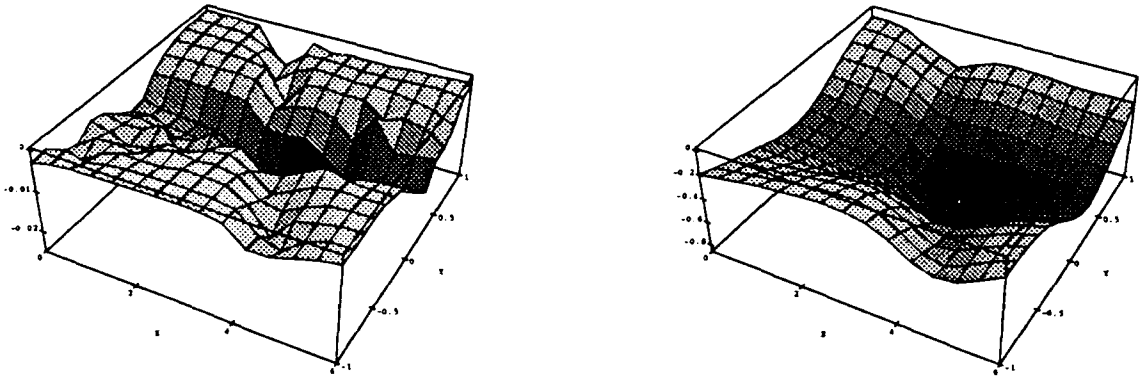


Figure 3: Deterministic annealing. (a) Objective function at $\sigma = .0364$ (very little smoothing). (b) Objective function at $\sigma = .300$. Smoothed; the minimum of this function provides a good starting point for minimizing (a).

derive new implementations of learning neural nets automatically, under all the constraints of limited hardware, communication and flexibility that have already been discussed. But in addition to these transformational considerations, we have a far more important suggestion for how to make substantial progress in learning. It involves the grammatical derivation of neural nets which will be introduced in section 2.3.

Global Optimization. Effective approximations to *global optimization* via neural nets look considerably more tractable since the recent flurry of research in deterministic annealing methods [8, 9, 10], none of which we were responsible for. These methods smooth a bumpy objective function and are often clearly automatable as objective function transformations, in which some of the constraints in a constrained optimization problem can be imposed exactly rather than by penalty terms. An interesting variety of new neural net transfer functions and dynamical systems result, and we continually find more by working on neural networks for computer vision [11]. Another approach to removing local minima will be described in the next section.

One example of the smoothing effect of deterministic annealing is our registration network for line-segment images [2, 12], in which the objective is a bumpy function of a global dispacement $\Delta \mathbf{x}$. The objective and a smoothed version are shown in figure 3.

Summary of Algebraic Transformations. The algebraic transformations above are far from a complete set that would suffice for a mathematical compiler, since some are still under development and

we have not yet tried very many target architectures or input neural nets, and since we have not yet fully automated the procedure. Also there is an opportunity, demonstrated by the log and exponential transformations available for analog CMOS, to build a library of technology-specific tricks to minimize implementation costs in important target technologies. But the transformations we studied do show that most of the main engineering issues that generate hardware diversity and that a compiler would have to resolve in translating objective functions to neural net implementations fall within the purview of algebraic transformations.

2.2 Software Engineering

Assertion 2. We have been accumulating the research in a neural simulator. It could be expanded into a semi-automatic compiler: a neural net design and implementation environment based on mathematical methods.

Algebraic Data Stucture. Our present simulator is based on the algebra of objective functions. The data structure holding the neural net is itself regarded (and implemented) as a large objective function in which synapses are monomial summands. Constraints can be added to the net via quadratic penalty functions (e.g. expanded out to make many new synapses on existing neurons) or lagrange multiplier neurons [13]. Optimization methods such as gradient descent, conjugate gradient, deterministic annealing and multiscale methods are packaged as "optimizers" which could be applied to a variety of optimization problems, both in learning and running a network. In this way optimization is modularized.

Class Libraries. Likewise "container classes" and a neuron indexing scheme modularize our data structure code, and the use of InterViews (a C++ interface to X-windows) modularizes a graphical user interface. We are experimenting with parallel implementations using Linda, a portable programming language extension. This is currently done with "fragment" objects which implement a kind of domain decomposition for neural nets. We have used the simulator mostly for problems in high-level vision, a rich domain of considerable interest in its own right. The simulator is in C++ and benefits from object-oriented design; in particular the "simulator" is not so much a single program as it is a reusable class library for neural simulation. It is possible that in the future we will also do object-oriented programming in CLOS (the Common Lisp Object System) or even Mathematica for symbolic algebra manipulation.

Towards a Compiler. In order to expand this software into a semi-automatic compiler it would be necessary to make it a bit more like a computer algebra system: more interactive and with more syntactic forms recognized. Indeed the compiler probably should be prototyped within an existing computer algebra system such as Mathematica. For example here is transformation Rule 1.1, $\frac{1}{2}X^2 \rightarrow X\sigma - \frac{1}{2}\sigma^2$. superficially translated into Mathematica:

```
(* Initialize the list of reversed neurons *)
ReversedNeurons = {}
```

```
(* We have to remove protection from the Power function so that we
   can attach a new transformation. The transformation will only
   be used when the expression x is more complex than a single number
   or variable. *)
Unprotect[Power]
x_^2 := QuadraticTransform[x] /; !AtomQ[x]
Protect[Power]

(* A new symbol is created with Unique and added to the list of reversed
   neurons. The final line is the transformed expression. *)
QuadraticTransform[x_] := Block[{usim = Unique["sigma"]},
        AppendTo[ReversedNeurons, usim];
        2 x usim - usim^2]
```

and here is a simple example of its use:

```
In[1]:= <<rule1.1.m

In[2]:= (x+y)^2       (* x+y can be transformed *)

                2
Out[2]= -sigma1  + 2 sigma1 (x + y)
```

The resulting expression could then be exported to a neural simulator or other implementation code. For the full compiler system, graphical tools analogous to standard circuit design tools would be included. For example each valid transformation rule could have its own pop-up window by which a human user could direct or modulate its application. Such interfaces have become far easier to construct in the last few years due to tools like InterViews and its interactive builder of user interfaces, IBuild. Gradually one could develop even greater automation, in which the choice of transformation rule and its locus of action is also automated.

Software Summary The purpose of such software engineering is not to develop a product but to demonstrate the feasability of a new kind of software tool for neural networks, based on serious mathematical methods which can be encapsulated as algebraic transformations, and to support research into both neural net design and implementation.

2.3 Mathematical Notation for Complex Problem Domains

Assertion 3. We have developed a mathematical notation (not yet a formal language) for describing complex problem domains in terms of constrained optimization problems. The optimization problems can be solved by neural nets as described in Assertion 1. Our notation involves objective functions, L-system grammars, and maps between such objects.

How to Compose Diverse Objective Functions? So far we have described the research situation and opportunities as if it were always not only possible but even easy to formulate neural net applications as constrained optimization problems. Unfortunately our experience is that it is generally possible but not trivial to get relatively compact problem specifications this way. We now perceive a need for a coarser level of structure - a symbolic and expressive programming language - which can "compose" or glue together many individual objective functions and algebraic constraints describing different aspects of a problem into a single modular problem description. Yet we do not want to give up the unique advantages (learning, circuit implementation) that neural nets gained by leaving traditional programming languages behind. *Recognizing and resolving this conflict is a major result of our research of the past few years.*

Grammars Can Compose Objectives. Our solution comes from the world of L-systems, parallel grammars originally introduced by Lindenmayer [14, 15] to describe plant growth. We introduce grammars (see Box 1) whose production rules are each governed by a Boltzmann probability distribution (i.e. by an algebraic objective function) which specifies when and how the rule may fire. Ordinary L-systems are attractive for physically-based computation because they can directly model highly parallel dynamical systems and, with just one rule, fractal growth processes. L-systems augmented with objective functions inherit, for each grammar rule, all of the power of constrained optimization we have advertised and exploited in vision. A grammar with many rules can correctly compose many diverse objective functions and create one organized problem specification. We have detailed a number of applications of such objective function grammars to vision in [12], including the transformational derivation of neural nets as in equation 1. Generally, one could say that these applications demonstrate the use of grammars in stating and solving visual pattern recognition problems. This is especially true where other neural methods fail: in making use of regularities which are abstract and remote from the pixel level. Another application of connectionist grammars, to the modelling of biological development, is described in [16].

Grammars support systems integration. One of the main sources of hardware diversity in artificial neural network design is the way in which a neural net is to be integrated with the rest of a computer system. Algebraic transformations were proposed for connecting up different hardware modules in section 2.1. At the software level, almost all software architectures will be far more easily integrated with a grammar than with a neural net, since computer languages of all sizes are fundamental in computer science and practice. In connectionist grammars we have a natural way to mix grammars with neural nets, which we think will support the integration of neural nets into generic computer systems.

Learning in Grammars. Connectionist grammars raise a fundamental new opportunity in learning. The reason is that machine learning, including neural net learning, is fundamentally dominated by

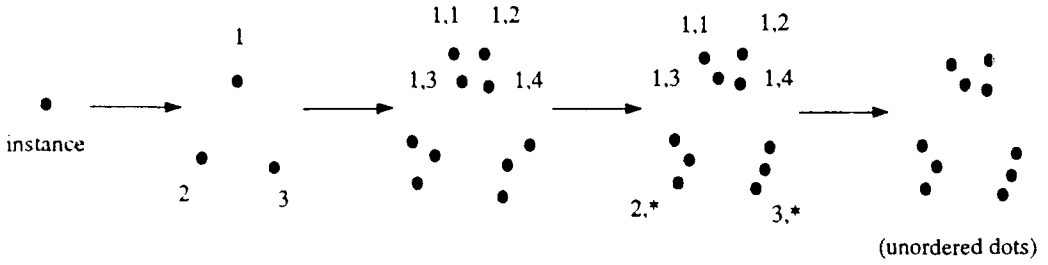
Box 1. Connectionist Grammars.

Consider an object with a hierarchical decomposition into parts, with internal degrees of freedom describing the relative positions of the parts. For random dot features, the resulting images will generally be clusters of dots with unpredictable jitter of both the dot and the cluster positions. A model of such an object is given by this grammar:

model locations	Γ^0 :	root	$\rightarrow$ instance of model α at $\mathbf{x}$
		$E_0(\mathbf{x})$	$= \frac{1}{2\sigma_r^2}\lvert\mathbf{x}\rvert^2$
jittered cluster locations	Γ^1 :	instance$(\alpha, \mathbf{x})$	$\rightarrow$ $\{\text{cluster}(\alpha, c, \mathbf{x}_c)\}$
		$E_1(\{\mathbf{x}_c\})$	$= \frac{1}{2\sigma_{cd}^2}\sum_c \lvert\mathbf{x}_c - \mathbf{x} - \mathbf{u}_c^\alpha\rvert^2$, where $< \mathbf{u}_c^\alpha >_c = 0$
jittered dot locations	Γ^2 :	cluster$(\alpha, c, \mathbf{x}_c)$	$\rightarrow$ $\{\text{dot}(c, m, \mathbf{x}_{cm})\}$
		$E_2(\{\mathbf{x}_{cm}\})$	$= \frac{1}{2\sigma_{jt}^2}\sum_m \lvert\mathbf{x}_{cm} - \mathbf{x}_c - \mathbf{u}_{cm}^\alpha\rvert^2$, where $< \mathbf{u}_{cm}^\alpha >_m = 0$
scramble all dots	Γ^3 :	$\{\text{dot}(c, m, \mathbf{x}_{cm})\}$	$\rightarrow$ $\{\text{imagedot}(\mathbf{x}_i = \sum_{cm} P_{cm,i}\mathbf{x}_{cm})\}$
		$E_3(\{\mathbf{x}_i\})$	$= -\log \prod_i \delta(\mathbf{x}_i - \sum_{cm} P_{cm,i}\mathbf{x}_{cm})$ where $\sum_i P_{m,i} = 1 \ \wedge \ \sum_m P_{m,i} = 1$

(3)

which is illustrated below:



The corresponding probability distribution is:

$$\Pr^3(\alpha, \mathbf{x}, \{\mathbf{x}_c\}, \{\mathbf{x}_i\}) = \frac{1}{A}\left(\frac{1}{\sqrt{2\pi}\sigma_r}\right)^2\left(\frac{1}{\sqrt{2\pi}\sigma_{cd}}\right)^{2C}\left(\frac{1}{\sqrt{2\pi}\sigma_{jt}}\right)^{2N}\sum_{\left\{\begin{array}{c}\text{P}\mid \text{P is a}\\ \text{permutation}\end{array}\right\}} \tag{4}$$

$$e^{-\sum_{cm,i} P_{cm,i}\left(\frac{1}{2N\sigma_r^2}\lvert\mathbf{x}\rvert^2 + \frac{C}{2N\sigma_{cd}^2}\lvert\mathbf{x}_c - \mathbf{x} - \mathbf{u}_c^\alpha\rvert^2 + \frac{1}{2\sigma_{jt}^2}\lvert\mathbf{x}_i - \mathbf{x}_c - \mathbf{u}_{cm}^\alpha\rvert^2\right)}$$

where C is the number of clusters and N/C is the number of dots in each cluster. From this expression, one can derive a neural net to recognize which model is present in the final image.

issues of representation: what are the input and output representations, and what kinds of internal representations can the learner actually create? Yet grammars offer representational flexibility approaching to that of conventional symbolic programming languages, in a quantitative and probabilistic context that makes scoring functions continuous and makes gradient-descent training possible. In other words, the best features of AI languages (flexible representation) and neural net parameterizations (learning and generalization) are combined to make a mathematical object (the connectionist grammar) for which both the human designer and the learning dynamics can deeply influence and vary the representations used.

Our proposed methods for learning a grammar have been spelled out in the last section of [12], from which we now quote. "Three possible methods for learning a grammar are suggested here. They all assume that learning the grammar can be expressed as tuning its parameters, as is the case for unstructured neural networks. First, the kind of grammar we have been studying could be augmented with an initial set of 'metagrammar' rules, which randomly choose the parameters of the permanent models and then generate many images by the usual grammar. The task of inferring the permanent models' parameters is just another Bayesian inference problem, stretched out over many images. Second, one could minimize the Kullback information [17] between the probability distributions of an unknown grammar, images from which the perceiver sees, and a parameterized grammar. This algorithm would be similar to the 'Boltzmann machine' for neural network learning [18]. Finally one could look for *clusters* in model space by defining a distance 'metric' D between images and mathematically projecting it back though the grammar." (There follows a general-purpose candidate D.) We propose to continue in these directions, which may not all be very different from each other.

Variable-Binding. A standard problem in appying neural nets to symbolic reasoning, as might be required in high-level vision or in language processing, is their limited ability to bind varibles within a local context. Such limitations on the expressiveness of neural nets are burdensome for programming or designing neural nets. We showed one optimization-based way to overcome these limitations in [19], where we translate a simple frame-based language and its valid deductions into a neural network architecture closely related to our graph-matching architecture for high-level vision nets [20].

Maps Between Grammars. A connectionist grammar is built out of objective functions and would therefore lie above optimization in a conventional layered-system diagram of the modelling language we are discussing. Likewise we now suggest a further level of abstraction beyond grammars: maps between grammars, for example the map by which one grammar is optimized to approximate another one which differs in number of rules or other properties. (Figure 4 is the corresponding layered-system diagram.) Other important maps between grammars could be defined, but this level of abstraction is beyond the grasp of our research so far.

Programming Language Research. In this section, in contrast to section 2.1, we have mainly discussed programming language issues rather than prospects for new mathematical methods. In a research program to develop mathematical methods, why is this necessary? The answer is that our proposed language for stating and solving application problems with neural nets must be both implementable and adequately expressive, as must any other computer language. Both criteria substantially

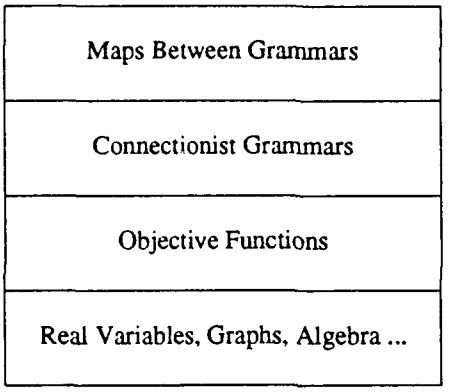


Figure 4: Components of a layered modelling language.

constrain the language, and in this way the expressiveness considerations typical of programming language research come to influence what implementation questions arise - i.e. what problems we will be called upon to solve with new mathematical methods.

Of course there also exist less well developed and nonquantitative branches of mathematics - such as logic and category theory - which pertain *directly to the expressiveness constraints* on a programming language. It is likely that connectionist grammars will develop in this direction to some extent.

3 Abstracts of works supported by AFOSR-88-0240

Eric Mjolsness and Willard L. Miranker, "A Lagrangian Approach to Fixed Points", Neural Information Processing Systems 3:

We present a new way to derive dissipative, optimizing dynamics from the Lagrangian formulation of mechanics. It can be used to obtain both standard and novel neural net dynamics for optimization problems. To demonstrate this we derive standard descent dynamics as well as nonstandard variants that introduce a computational attention mechanism.

Eric Mjolsness, Anand Rangarajan, and Charles Garrett, "A Neural Net for Reconstruction of Multiple Curves with a Visual Grammar", 1991 International Joint Conference on Neural Networks, Seattle:

We derive a neural net for reconstructing a set of curves from ungrouped dot locations. The network performs Bayesian inference on a *visual grammer*, which serves as probabilistic model of the image formation process, by means of quadratic matching objective function.

Eric Mjolsness, "Bayesian Inference on Visual Grammars by Neural Nets that Optimize", Technical Report YALEU/DCS/TR854, May 1991:

We exhibit a systematic way to derive neural nets for vision problems. It involves formulating a vision problem as Bayesian inference or decision on a comprehensive model of the visual domain given by a probabilistic *grammar*. A key feature of this grammar is the way in which it eliminates model information, such as object labels, as it produces an image; correspondance problems and other noise removal tasks result. The neural nets that arise most directly are generalized assignment networks. Also there are transformations which naturally yield improved algorithms such as correlation matching in scale space and the Frameville neural nets for high-level vision. Networks derived this way generally have objective functions with spurious local minima; such minima may commonly be avoided by dynamics that include deterministic annealing, for example recent improvements to Mean Field Theory dynamics. The grammatical method of neural net design allows domain knowledge to enter from all levels of the grammar, including "abstract" levels remote from the final image data, and may permit new kinds of learning as well.

Eric Mjolsness and Charles Garrett, "Algebraic Transformations of Objective Functions", Neural Networks, vol.3, pp 651-669, 1990:

Many neural networks can be derived as optimization dynamics for suitable objective functions. We show that such networks can be designed by repeated *transformations* of one objective into another with the same fixpoints. We exhibit a collection of algebraic transformations which reduce network cost and increase the set of objective functions that are neurally implementable. The transformations include simplification of products of expressions, functions of one or two expressions, and sparse matrix products (all of which may be interpreted as Legendre transformations); also the minimum and

maximum of a set of expressions. These transformations introduce new interneurons which force the network to seek a saddle point rather than a minimum. Other transformations allow control of the network dynamics, by reconciling the Lagrangian formalism with the need for fixpoints. We apply the transformations to simplify a number of structured neural networks, beginning with the standard reduction of the winner-take-all network from $\mathcal{O}(N^2)$ connections to $\mathcal{O}(N)$. Also susceptible are inexact graph-matching, random dot matching, convolutions and coordinate transformations, and sorting. Simulations show that fixpoint-preserving transformations may be applied repeatedly and elaborately, and the example networks still robustly converge.

Eric Mjolsness, Charles Garrett, and Willard L. Miranker, "Multiscale Optimization in Neural Nets", IEEE Transactions on Neural Networks, vol. 2, no. 2, March 1991:

One way to speed up convergence in a large optimization problem is to introduce a smaller, approximate version of the problem at a coarser scale and to alternate between relaxation steps for the fine-scale and the coarse-scale problems. We exhibit such an optimization method for neural networks governed by quite general objective functions. At the coarse scale there is a smaller approximating neural net which, like the original net, is nonlinear and has a nonquadratic objective function. The transitions and information flow from fine to coarse scale and back do not disrupt the optimization, and the user need only specify a partition of the original fine-scale variables. Thus the method can be applied easily to many problems and networks. We show positive experimental results including cost comparisons.

P. Anandan, Stanley Letovsky, and Eric Mjolsness, "Connectionist Variable-Binding By Optimization", August 1989 Cognitive Science conference proceedings.

Symbolic AI systems based on logical or frame languages can easily perform inferences that are still beyond the capabilities of most connectionist networks. This paper presents a strategy for implementing in connectionist networks the basic mechanisms of variable binding, dynamic frame allocation and equality that underlie mnay of the types of inferences commonly handled by frame systems, including inheritance, subsumption and abductive inference. The paper describes a scheme for translating frame definitions in a simple frame language into objective functions whose minima correspond to partial deductive closures of the legal inferences. The resulting constrained optimization problem can be viewed as a specification for a connectionist network.

Eric Mjolsness David H. Sharp and John Reinitz, "A Connectionist Model of Development", Journal of Theoretical Biology, v 152, pp. 429-453, 1991:

We present a phenomenological modeling framework for development. Our purpose is to provide a systematic method for discovering and expressing correlations in experimental data on gene expression and other developmental processes. The modeling framework is based on a connectionist or "neural net" dynamics for biochemical regulators, coupled to "grammatical rules" which describe certain features of the birth, growth, and death of cells, synapses and other biological entities. We outline how spatial geometry can be included, although this part of the model is not complete. As an example of

the application of our results to a specific biological system, we show in detail how to derive a rigorously testable model of the network of segmentation genes operating in the blastoderm of *Drosophila*. To further illustrate our methods, we sketch how they could be applied to two other important developmental processes: cell cycle control and cell-cell induction. We also present a simple biochemical model leading to our assumed connectionist dynamics which shows that the dynamics used is at least compatible with known chemical mechanisms.

4 References

References

[1] E. Mjolsness, "Mathematical methods for the implementation of neural networks." Renewal proposal from Yale University to the Air Force Office of Scientific Research, September 1991.

[2] E. Mjolsness and C. Garrett, "Algebraic transformations of objective functions," *Neural Networks*, vol. 3, pp. 651–669, 1990.

[3] C. A. Mead, *Analog VLSI and Neural Systems*, ch. 6. Addison Wesley, 1989.

[4] E. Mjolsness and C. Garrett, "Unpublished results." Pilot study on an Encore Multimax., 1991.

[5] E. Mjolsness and W. L. Miranker, "A Lagrangian approach to fixed points," in *Neural Information Processing Systems 3* (R. P. Lippmann, J. E. Moody, and D. S. Touretzky, eds.), Morgan Kaufmann, 1991.

[6] E. Mjolsness, C. D. Garrett, and W. L. Miranker, "Multiscale optimization in neural nets," *IEEE Transactions on Neural Networks*, vol. 2, March 1991.

[7] J. Hopfield, "Private communication." ., 1986.

[8] P. D. Simic, "Statistical mechanics as the underlying theory of 'elastic' and 'neural' optimization," *Network: Computation in Neural Systems*, vol. 1, pp. 89–103, January 1990.

[9] C. Peterson and B. Soderberg, "A new method for mapping optimization problems onto neural networks," *International Journal of Neural Systems*, vol. 1, no. 3, 1989.

[10] A. L. Yuille, "Generalized deformable models, statistical physics, and matching problems," *Neural Computation*, vol. 2, no. 1, pp. 1–24, 1990.

[11] E. Mjolsness, A. Rangarajan, and C. Garrett, "A neural net for reconstruction of multiple curves with a visual grammar," in *International Joint Conference on Neural Networks*, pp. I–615 to I–620, IEEE, July 1991.

[12] E. Mjolsness, "Bayesian inference on visual grammars by neural nets that optimize," Tech. Rep. YALEU/DCS/TR-854, Yale Computer Science Department, May 1991. Currently available from the 'Neuroprose' computer archive.

[13] J. C. Platt and A. H. Barr, "Constrained differential optimization," in *Neural Information Processing Systems* (D. Z. Anderson, ed.), American Institute of Physics, 1987.

[14] A. Lindenmayer, "Mathematical models for cellular interaction in development, Parts I and II," *Journal of Theoretical Biology*, vol. 18, pp. 280–315, 1968.

[15] P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants.* Springer-Verlag New York, 1990.

[16] E. Mjolsness, D. H. Sharp, and J. Reinitz, "A connectionist model of development," *Journal of Theoretical Biology*, 1991. In press. Also available as Yale Computer Science technical report YALEU/DCS/796, June 1990.

[17] S. Kullback, *Information Theory and Statistics.* Wiley, New York, 1959.

[18] G. E. Hinton and T. J. Sejnowski, "Learning and relearning in Boltzmann machines," in *Parallel Distributed Processing* (D. E. Rumelhart and J. L. McClelland, eds.), ch. 7, MIT Press, 1986.

[19] P. Anandan, S. Letovsky, and E. Mjolness, "Connectionist variable-binding by optimization," August 1989. Cognitive Science society conference proceedings.

[20] E. Mjolsness, G. Gindi, and P. Anandan, "Optimization in model matching and perceptual organization," *Neural Computation*, vol. 1, 1989.

A Appendix: Mathematica Code for Algebraic Transformations

Following the example of section 2.2, we present some more extensive Mathematica code (by C. Garrett) which uses symbolic algebra to implement the objective function transformations of [2] (included as Appendix B). This code is the start of the front-end portion of a neural network compiler.

The C++ back end has also been started but obviously it would be too long to include as an appendix. To make the present appendix self-contained, we demonstrate the syntax appropriate for optimizing the objective functions entirely within Mathematica, as well as that for using the much more efficient back end.

File "nemesis-examples"

```
A Description of Nemesis for Mathematica

7 / 28 / 92


    The Nemesis.m package contains Mathematica code for constructing and
transforming objective functions. We will describe how the code works,
focusing on several examples. The description will break down into 3
major parts, index domains, objective function transformations,
and how to run networks.


Index Domains

    An index domain is a set of values that an index may have. For our
purposes, an index domain will be either an interval, or a disjoint
union or cross product of 2 other domains. Here are some examples of
valid and invalid index domains, in Mathematica-ese, and English.

IndexDomain[1, 5]       A single index with values from 1 to 5, inclusive.
IndexDomain[-2, n] A single index with values from -2 to n.
DisjointUnion[IndexDomain[0, 10], IndexDomain[20, 30]]
A single index with values from 0 to 10 and from 20 to
30.
DisjointUnion[IndexDomain[1, 5], IndexDomain[1, 5]]
A single index which takes on values from 1 to 5 twice.
CrossProductDomain[IndexDomain[1, 5], IndexDomain[1, 4]]
A two part index which takes on values from {1, 1} to
{5, 4}.

If these function names seem too long, remember that you can use Mathematica
to equate shorter names to these, for instance CPD = CrossProductDomain.
We use long names simply to be unambiguous.
```

That accounts for the representation of index domains, but how are they used? The functions SumOver and TableOf interpret the index domains. For example SumOver can break down a complicated index domain into a sum over intervals like this:

```
In[4]:= BIGD = CrossProductDomain[DisjointUnion[IndexDomain[1, 5],
                                                IndexDomain[11, 15]],
                                  DisjointUnion[IndexDomain[6, 10],
                                                IndexDomain[16, 20]]]

In[5]:= SumOver[x[i][j], {i, j, BIGD}]

Out[5]= SumOver[x[i][j], {i, 1, 5}, {j, 6, 10}] +

>     SumOver[x[i][j], {i, 1, 5}, {j, 16, 20}] +

>     SumOver[x[i][j], {i, 11, 15}, {j, 6, 10}] +

>     SumOver[x[i][j], {i, 11, 15}, {j, 16, 20}]
```

Here you can see that the domain BIGD has 4 separate parts, each of which has 2 dimensions, and the SumOver function breaks the domain down into its separate parts.

Objective Function Transformations

Objective function transformations take a part of an expression and replace with a new expression that shares the same fixed points, but have fewer multiplications. In the process we introduce new variables, which may act to minimize or maximize the objective function. The transformations are invoked by applying one of the transform functions to an objective, like this:

```
In[2]:= SquareTransform[SumOver[x[i], {i, 5}]^2]

                2
Out[2]= -sigma1  + 2 sigma1 SumOver[x[i], {i, 5}]

In[3]:= MultiplyTransform[(a + b) (c + d)]

              2         2
        omega1    sigma2
Out[3]= ------- - ------- + (c + d) (-omega1 + sigma2) +
           2         2
```

```
                                      2
                                   tau1
>    (a + b) (sigma2 - tau1) + -----
                                    2
```

The new variables always have unique names, so that they do not conflict with other variables in the objective function. Also the names of the variables are placed into lists called ReversedNeurons and FastNeurons, so that you can tell the optimizers which way to move them.

```
In[4]:= ReversedNeurons

Out[4]= {sigma1, sigma2}

In[5]:= FastNeurons

Out[5]= {tau1, omega1}
```

Running Networks (within Mathematica)

We have written some optimizers which do not require any backend. They are defined in the file Descent.m. The next section is a brief demonstration of how you can use these optimizers.

First, we start up Mathematica, and read in the package *Descent.m*. Nemesis.m will automatically be loaded when we read in Descent.m.

```
In[1]:= <<Descent.m
```

Descent.m defines two functions, GradDesc and SaddlePoint. GradDesc takes an objective function as its argument. It gives each variable in the objective a random initial value, and then performs gradient descent to find a local minimum of the objective. Here is a simple example:

```
In[2]:= GradDesc[a^2]
{0.21235}
{0.208103}
{0.203941}
   ...          ( many lines of output are skipped here )
{0.0038888}
{0.00381102}
{0.0037348}

Out[2]= {a -> 0.0037348}
```

GradDesc has driven the variable 'a' close to its minimum value of 0. Remember that if you run the same network, you will see a slightly different

result due to the random initial conditions. The final output of GradDesc is a rule which you may use to replace the symbol 'a' by its nearly optimal value. In order to calculate the value of the objective function, you can say:

```
In[3]:= a^2 /. Out[2]

Out[3]= 0.0000139487
```

Now let's try a more complicated objective function which is defined in Nemesis.m. Here is the definition of Match.

```
************  code from Nemesis.m  *******

Match[x_, m_, n_] :=
SumOver[(SumOver[x[i][j], {j, n}] - 1)^2, {i, m}] +
SumOver[(SumOver[x[i][j], {i, m}] - 1)^2, {j, n}] +
SumOver[x[i][j] (1 - x[i][j]), {i, m}, {j, n}] +
SumOver[Potential[x[i][j]], {i, m}, {j, n}]

************  end of code ****************
```

Match returns an objective function which implements soft winner-take-all constraints on the rows and columns of a rectangular matrix. We can pass this directly to the GradDesc function. Since this network is fairly large, we will let it run for 500 steps.

```
In[6]:= GradDesc[Match[M, 5, 5], 500]
{0.282214, 0.268753, 0.212327, 0.231943, 0.274453, 0.279035, 0.146517,

>   0.256717, 0.196147, 0.137234, 0.231377, 0.225728, 0.246399, 0.225022,

>   0.105801, 0.213226, 0.278814, 0.15593, 0.235356, 0.275095, 0.295193,

>   0.1642, 0.27119, 0.219052, 0.146877}


            ( many steps removed )

{0.0424614, 0.0471485, 0.0439794, 0.040272, 0.866579, 0.0424675, 0.0471054,

>   0.868152, 0.039981, 0.043376, 0.0391433, 0.0425031, 0.0399867, 0.916185,

>   0.0400393, 0.045329, 0.832142, 0.04626, 0.0421782, 0.0469582, 0.884906,

>   0.0447747, 0.0422182, 0.0390809, 0.0423163}

Out[6]= {M[1][1] -> 0.0424614, M[1][2] -> 0.0471485, M[1][3] -> 0.0439794,

>    M[1][4] -> 0.040272, M[1][5] -> 0.866579, M[2][1] -> 0.0424675,
```

```
>    M[2][2] -> 0.0471054, M[2][3] -> 0.868152, M[2][4] -> 0.039981,

>    M[2][5] -> 0.043376, M[3][1] -> 0.0391433, M[3][2] -> 0.0425031,

>    M[3][3] -> 0.0399867, M[3][4] -> 0.916185, M[3][5] -> 0.0400393,

>    M[4][1] -> 0.045329, M[4][2] -> 0.832142, M[4][3] -> 0.04626,

>    M[4][4] -> 0.0421782, M[4][5] -> 0.0469582, M[5][1] -> 0.884906,

>    M[5][2] -> 0.0447747, M[5][3] -> 0.0422182, M[5][4] -> 0.0390809,

>    M[5][5] -> 0.0423163}
```

The final answer is close to a permutation network. You can see the structure more clearly by printing out a table of M's in MatrixForm.

```
In[7]:= MatrixForm[Table[M[i][j], {i, 5}, {j, 5}] /. Out[6]]

Out[7]//MatrixForm= 0.0424614   0.0471485   0.0439794   0.040272    0.866579

                    0.0424675   0.0471054   0.868152    0.039981    0.043376

                    0.0391433   0.0425031   0.0399867   0.916185    0.0400393

                    0.045329    0.832142    0.04626     0.0421782   0.0469582

                    0.884906    0.0447747   0.0422182   0.0390809   0.0423163
```

Now, we will try using an objective function transformation on the Match network. As a first step, let's look at the structure of the expression generated by Match[M, 5, 5].

```
In[9]:= func = Match[M, 5, 5]

                                                    2
Out[9]= SumOver[(-1 + SumOver[M[i][j], {i, 5}]) , {j, 5}] +

                                                    2
>    SumOver[(-1 + SumOver[M[i][j], {j, 5}]) , {i, 5}] +

>    SumOver[Potential[M[i][j]], {i, 5}, {j, 5}] +

>    SumOver[(1 - M[i][j]) M[i][j], {i, 5}, {j, 5}]
```

We can use the function SquareTransform to produce a new objective function with terms of the form x^2 transformed into 2 x sigma - simga^2.

```
In[11]:= func = SquareTransform[func]

                              2
Out[11]= SumOver[-sigma1[j]  + 2 sigma1[j] (-1 + SumOver[M[i][j], {i, 5}]),

                                    2
>     {j, 5}] + SumOver[-sigma2[i]  +

>      2 sigma2[i] (-1 + SumOver[M[i][j], {j, 5}]), {i, 5}] +

>    SumOver[0.1 (-Abs[-0.5 + M[i][j]] - 0.5 Log[0.5 - Abs[-0.5 + M[i][j]]]),

>     {i, 5}, {j, 5}] + SumOver[(1 - M[i][j]) M[i][j], {i, 5}, {j, 5}]
```

In this objective function, sigma1[j] governs the column constraints, and sigma2[i] governs the row constraints. Since the new sigma neurons act to maximize the objective function, we cannot use the same GradDesc optimizer which sends all neurons downhill. Instead we use the optimizer SaddlePoint, and pass it a list of the reversed neurons. Conveniently, the SquareTransform function appends the names of all the new reversed neurons it creates to the list ReversedNeurons.

```
In[12]:= SaddlePoint[func, ReversedNeurons, 500]
{0.112055, 0.287099, 0.235514, 0.23606, 0.129563, 0.110942, 0.282688,

>   0.16964, 0.175481, 0.175472, 0.259398, 0.170251, 0.282214, 0.268753,

>   0.206656, 0.279856, 0.106098, 0.282597, 0.21139, 0.232409, 0.26157,

>   0.135819, 0.10515, 0.21158, 0.104505, 0.199769, 0.105801, 0.225568,

>   0.165773, 0.141062, 0.2621, 0.152875, 0.289367, 0.245156, 0.26151}

        ( many steps removed )

{0.0404085, 0.0404085, 0.0452528, 0.881893, 0.0404189, 0.0374939, 0.919705,

>   0.0408759, 0.0403384, 0.0375015, 0.0408049, 0.0408049, 0.875959,

>   0.0453532, 0.0408156, 0.0375015, 0.0375015, 0.0408868, 0.0403486,

>   0.919624, 0.919705, 0.0374939, 0.0408758, 0.0403384, 0.0375015, 0.076609,

>   0.0766089, 0.0343611, 0.0401126, 0.076517, 0.0398592, 0.076615,

>   0.0345962, 0.0765232, 0.0766152}

Out[12]= {M[1][1] -> 0.0404085, M[1][2] -> 0.0404085, M[1][3] -> 0.0452528,

>    M[1][4] -> 0.881893, M[1][5] -> 0.0404189, M[2][1] -> 0.0374939,
```

```
>    M[2][2] -> 0.919705, M[2][3] -> 0.0408759, M[2][4] -> 0.0403384,
>    M[2][5] -> 0.0375015, M[3][1] -> 0.0408049, M[3][2] -> 0.0408049,
>    M[3][3] -> 0.875959, M[3][4] -> 0.0453532, M[3][5] -> 0.0408156,
>    M[4][1] -> 0.0375015, M[4][2] -> 0.0375015, M[4][3] -> 0.0408868,
>    M[4][4] -> 0.0403486, M[4][5] -> 0.919624, M[5][1] -> 0.919705,
>    M[5][2] -> 0.0374939, M[5][3] -> 0.0408758, M[5][4] -> 0.0403384,
>    M[5][5] -> 0.0375015, sigma1[1] -> 0.076609, sigma1[2] -> 0.0766089,
>    sigma1[3] -> 0.0343611, sigma1[4] -> 0.0401126, sigma1[5] -> 0.076517,
>    sigma2[1] -> 0.0398592, sigma2[2] -> 0.076615, sigma2[3] -> 0.0345962,
>    sigma2[4] -> 0.0765232, sigma2[5] -> 0.0766152}

In[13]:= MatrixForm[Table[M[i][j], {i, 5}, {j, 5}] /. Out[12]]

Out[13]//MatrixForm= 0.0404085   0.0404085   0.0452528   0.881893    0.0404189
                     0.0374939   0.919705    0.0408759   0.0403384   0.0375015
                     0.0408049   0.0408049   0.875959    0.0453532   0.0408156
                     0.0375015   0.0375015   0.0408868   0.0403486   0.919624
                     0.919705    0.0374939   0.0408758   0.0403384   0.0375015
```

This answer is also a good permutation matrix.

Running Networks (with the back end)

In order to use the C++ back end, you must install the routines which are in the file MathTree. These routines allow you to describe networks and optimizers, define reversed or fast neurons, initialize neuron values, and finally run networks.

```
In[14]:= Install["MathTree"]

Out[14]= LinkObject[MathTree, 1, 1]
```

We will continue with the permutation matrix example network that we used above, and this time we will write the objective function using an index

domain. The new function is called MatchDomain.

```
In[31]:= MatchDomain[x_, id_] :=
SumOver[(SumOver[x[i][j], {j, Dimension[2, id]}] - 1)^2,
  {i, Dimension[1, id]}] +
SumOver[(SumOver[x[i][j], {i, Dimension[1, id]}] - 1)^2,
  {j, Dimension[2, id]}] +
SumOver[x[i][j] (1 - x[i][j]), {i, j, id}] +
SumOver[Potential[x[i][j]], {i, j, id}]
```

This function will take an index domain with 2 dimensions, and impose winner-take-all constraints along each dimension. Now let's define an index domain and call MatchDomain to see what the objective function will look like.

```
In[32]:= CPD = CrossProductDomain[IndexDomain[1, 5], IndexDomain[1, 5]]

Out[32]= CrossProductDomain[IndexDomain[1, 5], IndexDomain[1, 5]]

In[33]:= MatchDomain[M, CPD]

                                                    2
Out[33]= SumOver[(-1 + SumOver[M[i][j], {i, 1, 5}]) , {j, 1, 5}] +

                                                  2
>    SumOver[(-1 + SumOver[M[i][j], {j, 1, 5}]) , {i, 1, 5}] +

>    SumOver[Potential[M[i][j]], {i, 1, 5}, {j, 1, 5}] +

>    SumOver[(1 - M[i][j]) M[i][j], {i, 1, 5}, {j, 1, 5}]
```

We now communicate the objective function to the back end program with the function SetUpNet, and choose a line minimization optimizer with SetUpOptimizer.

```
In[34]:= SetUpNet[MatchDomain[M, CPD]]

Variable M:
0.088342 0.113757 0.027181 0.094360 0.036566
0.194903 0.111318 0.085666 0.111402 0.103520
0.039403 0.137246 0.015866 0.092842 0.034000
0.024054 0.035861 0.139221 0.090609 0.033407
0.037842 0.016067 0.087616 0.051261 0.122438

Out[34]= 1

In[35]:= SetUpOptimizer[lm]
```

```
Using line minimizer

Out[35]= 1

Now we are ready to run the network with RunNet.

In[36]:= RunNet[]
Variable M:
0.179168 0.202559 0.130800 0.179705 0.137766
0.267729 0.175160 0.154529 0.172607 0.175494
0.138263 0.231581 0.138975 0.182257 0.140355
0.131180 0.132569 0.239219 0.179584 0.139649
0.137620 0.133204 0.185878 0.142404 0.224744

energy = 4.084525

 ( many steps of output removed )

energy = 1.384581

Variable M:
0.011132 0.011135 0.011132 0.985656 0.011132
0.985724 0.011134 0.011131 0.011133 0.011131
0.011129 0.985723 0.011129 0.011132 0.011129
0.011131 0.011134 0.985724 0.011133 0.011131
0.011131 0.011134 0.011131 0.011133 0.985724


Out[36]= {M[1][1] -> 0.0111322, M[1][2] -> 0.0111351, M[1][3] -> 0.0111322,

>    M[1][4] -> 0.985656, M[1][5] -> 0.0111322, M[2][1] -> 0.985724,

>    M[2][2] -> 0.0111339, M[2][3] -> 0.0111306, M[2][4] -> 0.0111333,

>    M[2][5] -> 0.0111306, M[3][1] -> 0.0111288, M[3][2] -> 0.985723,

>    M[3][3] -> 0.0111288, M[3][4] -> 0.0111316, M[3][5] -> 0.0111288,

>    M[4][1] -> 0.0111306, M[4][2] -> 0.0111339, M[4][3] -> 0.985724,

>    M[4][4] -> 0.0111333, M[4][5] -> 0.0111306, M[5][1] -> 0.0111306,

>    M[5][2] -> 0.0111339, M[5][3] -> 0.0111306, M[5][4] -> 0.0111333,

>    M[5][5] -> 0.985724}
```

The answer is a permutation matrix, as we expected. And now just to complete the demonstration, we will run the permutation matrix network with reversed neurons.

```
In[37]:= func = SquareTransform[MatchDomain[M, CPD]]

                                2
Out[37]= SumOver[-sigma1[j]  +

>       2 sigma1[j] (-1 + SumOver[M[i][j], {i, 1, 5}]), {j, 1, 5}] +

                          2
>     SumOver[-sigma2[i]  + 2 sigma2[i] (-1 + SumOver[M[i][j], {j, 1, 5}]),

>      {i, 1, 5}] + SumOver[Potential[M[i][j]], {i, 1, 5}, {j, 1, 5}] +

>     SumOver[(1 - M[i][j]) M[i][j], {i, 1, 5}, {j, 1, 5}]

In[38]:= SetUpNet[func]

Variable sigma2:
-0.233151 0.275137 -1.456376 -0.112794 -1.268687
Variable M:
0.194903 0.111318 0.085666 0.111402 0.103520
0.039403 0.137246 0.015866 0.092842 0.034000
0.024054 0.035861 0.139221 0.090609 0.033407
0.037842 0.016067 0.087616 0.051261 0.122438
0.155670 0.026563 0.064878 0.161136 0.069341

Variable sigma1:
0.558585 1.640131 -0.606858 -0.872527 -1.816334

Out[38]= 1

In[39]:= SetUpOptimizer[lm]

Using line minimizer

Out[39]= 1

In[40]:= SetUpReversedNeurons[ReversedNeurons]


Out[40]= 1

In[41]:= RunNet[]
Variable sigma2:
-0.393191 -0.680643 -0.676848 -0.684777 -0.522411
Variable M:
0.710856 0.678656 0.593971 0.537001 0.640001
0.758736 0.942796 0.947946 0.738820 0.839069
0.807958 0.859952 0.890754 0.733150 0.837591
0.764348 0.999758 0.825588 0.710739 0.895874
0.748781 0.769826 0.676887 0.713906 0.704011
```

```
Variable sigma1:
-0.548127 -0.672945 -0.606754 -0.492750 -0.637294
energy = -33.651328

( after many steps )


energy = 1.384579

Variable sigma2:
0.029720 0.029719 0.029244 0.029719 0.029719
Variable M:
0.011176 0.011173 0.011176 0.011176 0.985732
0.011176 0.011174 0.985732 0.011176 0.011176
0.011173 0.985541 0.011173 0.011173 0.011173
0.985732 0.011174 0.011176 0.011176 0.011176
0.011176 0.011174 0.011176 0.985732 0.011176

Variable sigma1:
0.029719 0.029243 0.029719 0.029719 0.029719

Out[41]= {sigma2[1] -> 0.02972, sigma2[2] -> 0.0297189,

>    sigma2[3] -> 0.0292441, sigma2[4] -> 0.0297189, sigma2[5] -> 0.0297189,

>    M[1][1] -> 0.0111758, M[1][2] -> 0.0111735, M[1][3] -> 0.0111758,

>    M[1][4] -> 0.0111758, M[1][5] -> 0.985732, M[2][1] -> 0.0111759,

>    M[2][2] -> 0.0111736, M[2][3] -> 0.985732, M[2][4] -> 0.0111759,

>    M[2][5] -> 0.0111759, M[3][1] -> 0.0111735, M[3][2] -> 0.985541,

>    M[3][3] -> 0.0111735, M[3][4] -> 0.0111735, M[3][5] -> 0.0111735,

>    M[4][1] -> 0.985732, M[4][2] -> 0.0111736, M[4][3] -> 0.0111759,

>    M[4][4] -> 0.0111759, M[4][5] -> 0.0111759, M[5][1] -> 0.0111759,

>    M[5][2] -> 0.0111736, M[5][3] -> 0.0111759, M[5][4] -> 0.985732,

>    M[5][5] -> 0.0111759, sigma1[1] -> 0.0297194, sigma1[2] -> 0.0292435,

>    sigma1[3] -> 0.0297194, sigma1[4] -> 0.0297194, sigma1[5] -> 0.0297194}
```

File "Nemesis.m"

```
WTA::usage =
    "WTA[x, n] returns a complete objective function for a winner
    take all net, involving variables x[1] through x[n]."

Match::usage =
    "Match[x, m, n] returns a complete objective function for a
    permutation matrix, involving variables x[1][1] through x[m][n]."

SquareTransform::usage =
    "SquareTransform[e] returns a copy of e, except that the terms
    of the form x^2 are replaced by 2 x sigma - sigma^2."

MultiplyTransform::usage =
    "MultiplyTransform[e] returns a copy of e, except that the terms
    of the form x y are replaced by x (sigma - tau) + y (sigma - omega)
    + 1/2 (-sigma^2 + tau^2 + omega^2)."

MultiplyTransform2::usage =
    "MultiplyTransform2[e] returns a copy of e, except that the terms
    of the form x y are replaced by 1/2 x (sigma - tau) + 1/2 y (sigma + tau)
    + 1/4 (-sigma^2 + tau^2)."

ExponentialTransform::usage =
    "ExponentialTransform[e] returns a copy of e, except that the terms
    of the form Exp[x] are replaced by (x + 1) sigma + sigma Log[sigma]."

XLogXTransform::usage =
    "XLogXTransform[e] returns a copy of e, except that the terms
    of the form Abs[x] Log[Abs[x]] are replaced by Abs[x] sigma -
    Exp[sigma]."

LogXTransform::usage =
    "LogXTransform[e] returns a copy of e, except that the terms
    of the form Log[Abs[x]] are replaced by x sigma - Log[Abs[sigma]]."

AbsPowerTransform::usage =
    "AbsPowerTransform[e] returns a copy of e, except that the terms
    of the form Abs[x]^p/p are replaced by x sigma -
    Abs[sigma]^(p/(p-1)) ((p-1)/p)."

SumOver::usage =
    "SumOver[e, {i, n}, {j, m}, ...] represents a sum over expression
    e, with summation indices i, j, ... just like the built in Sum
    function. The difference is that Mathematica will not try to
```

```
    expand SumOver the way it expands Sum."

TableOf::usage =
    "TableOf[e, {i, n}, {j, m}, ...] represents a list of expressions
    e, with indices i, j, ... just like the built in Table function.
    The difference is that Mathematica will not try to expand TableOf
    the way it expands Table."

IndexDomain::usage =
    "IndexDomain[l, h] is is a domain in which an index can take on
    values l, l+1, l+2, ..., h."

DisjointUnion::usage =
    "The DisjointUnion of 2, IndexDomains is a domain which
    contains all of the index values from each domain, and repeats
    those values which occur in more than both domains."

CrossProductDomain::usage =
    "The CrossProductDomain of 2 IndexDomains is a domain which
    contains one index value for each possible combination of index values
    in the original domains."

ReversedNeurons::usage =
    "The list ReversedNeurons contains the neurons introduced by objective
    function transformations which act to maximize the objective."

FastNeurons::usage =
    "The list FastNeurons contains the neurons introduced by objective
    function transformations which act to minimize the objective."

(* Some basic objective functions *)


WTA[x_, n_] :=
(SumOver[x[i], {i, n}] - 1)^2 +
SumOver[x[i] (1 - x[i]), {i, n}] +
SumOver[Potential[x[i], 0, 1], {i, n}]

Match[x_, m_, n_] :=
SumOver[(SumOver[x[i][j], {j, n}] - 1)^2, {i, m}] +
SumOver[(SumOver[x[i][j], {i, m}] - 1)^2, {j, n}] +
SumOver[x[i][j] (1 - x[i][j]), {i, m}, {j, n}] +
SumOver[Potential[x[i][j]], {i, m}, {j, n}]


(* Objective function transformations  *)

(* Notice the square transform only applies to non-Atomic expressions.
```

```
   Each transformation rule will be responsible for adding the neurons
   it creates to the lists ReversedNeurons or FastNeurons. *)

ReversedNeurons = {}

FastNeurons = {}

(* FreeIndices[x] returns the indices in expression x which are not
   bound by any SumOver expression in x itself. *)

FreeIndices[SumOver[x_, {i_, id_}]] :=
Complement[FreeIndices[x], {i}]

FreeIndices[SumOver[x_, {i_, j_Integer, id_}]] :=
Complement[FreeIndices[x], {i}]

FreeIndices[SumOver[x_, {i_, j_, id_}]] :=
Complement[FreeIndices[x], {i, j}]

FreeIndices[SumOver[x_, {i_, j_, k_, id__}]] :=
Complement[FreeIndices[x], {i, j, k}]

FreeIndices[SumOver[x_, {i_, j_, k_, l_, id__}]] :=
Complement[FreeIndices[x], {i, j, k, l}]

FreeIndices[Plus[x_, y_]] :=
Union[FreeIndices[x], FreeIndices[y]]

FreeIndices[Times[x_, y_]] :=
Union[FreeIndices[x], FreeIndices[y]]

FreeIndices[Power[x_, y_]] :=
Union[FreeIndices[x], FreeIndices[y]]

FreeIndices[Log[x_]] :=
FreeIndices[x]

FreeIndices[Exp[x_]] :=
FreeIndices[x]

FreeIndices[Abs[x_]] :=
FreeIndices[x]

FreeIndices[Potential[x_]] :=
FreeIndices[x]

FreeIndices[x_] :=
Block[{},
```

```
    If[NumberQ[x] || Head[x] == Symbol, Return[{}], 0];
    If[Head[x[[1]]] == Symbol, Return[Union[FreeIndices[Head[x]],
{x[[1]]}]], 0];
    FreeIndices[Head[x]]
]

AddIndices[s_, li_] :=
Block[{},
    If[Length[li] == 0, Return[s], 0];
    AddIndices[s[ li[[1]] ], Rest[li]]
]

SquareTransformRule = Power[x_, 2] :>
Block[{sig = Unique["sigma"], fi = FreeIndices[x]},
AppendTo[ReversedNeurons, sig];
2 x AddIndices[sig, fi] -
AddIndices[sig, fi]^2] /; !AtomQ[x]

SquareTransform[e_] :=
e /. SquareTransformRule

(* The multiplication transformation will only be applied when neither of
   the two expressions are simply numbers, and when neither contains any
   reversed neurons. The transformation introduces 3 new neurons, sigma,
   tau and omega.*)

MultiplyTransformRule = Times[x_, y_] :>
Block[{sigma = Unique["sigma"], tau = Unique["tau"],
                omega = Unique["omega"], fi = FreeIndices[x]},
                AppendTo[ReversedNeurons, sigma];
                AppendTo[FastNeurons, tau];
                AppendTo[FastNeurons, omega];
                x (AddIndices[sigma, fi] - AddIndices[tau, fi]) +
y (AddIndices[sigma, fi] - AddIndices[omega, fi]) -
AddIndices[sigma, fi]^2/2 + AddIndices[tau, fi]^2/2 +
AddIndices[omega, fi]^2/2] /;
FreeReversedQ[x] && FreeReversedQ[y] && !NumberQ[x] && !NumberQ[y]

(* FreeReversedQ returns True if the expression x does not contain any of
   the symbols on the list ReversedNeurons. It returns False otherwise *)
FreeReversedQ[x_] := Block[{i},
        For[i=1, i<=Length[ReversedNeurons], i++, If[FreeQ[x,
        ReversedNeurons[[i]]], , Return[False]]];
        Return[True]]

MultiplyTransform[e_] :=
```

```
e /. MultiplyTransformRule

(* The second multiplication transformation just introduces 2 new variables. *)

MultiplyTransformRule2 = Times[x_, y_] :>
Block[{sigma = Unique["sigma"], tau = Unique["tau"],
fi = FreeIndices[x]},
                AppendTo[ReversedNeurons, sigma];
                AppendTo[FastNeurons, tau];
                x (AddIndices[sigma, fi] - AddIndices[tau, fi])/2 +
y (AddIndices[sigma, fi] + AddIndices[tau, fi])/2
                - AddIndices[sigma, fi]^2/4 + AddIndices[tau, fi]^2/4] /;
FreeReversedQ[x] && FreeReversedQ[y] && !NumberQ[x] && !NumberQ[y]

MultiplyTransform2[e_] :=
e /. MultiplyTransformRule

ExponentialTransformRule = Power[E, x_] :>
Block[{sigma = Unique["sigma"], fi = FreeIndices[x]},
         AppendTo[ReversedNeurons, sigma];
         (x + 1) AddIndices[sigma, fi] -
AddIndices[sigma, fi] Log[AddIndices[sigma, fi]]]

ExponentialTransform[e_] :=
e /. ExponentialTransformRule

XLogXTransformRule = Abs[x_] Log[Abs[x_]] :>
Block[{sigma = Unique["sigma"], fi = FreeIndices[x]},
         AppendTo[ReversedNeurons, sigma];
         Abs[x] AddIndices[sigma, fi] - Exp[AddIndices[sigma, fi]]]

XLogXTransform[e_] :=
e /. XLogXTransformRule

LogXTransformRule = Log[Abs[x_]] :>
Block[{sigma = Unique["sigma"], fi = FreeIndices[x]},
         AppendTo[FastNeurons, sigma];
         x AddIndices[sigma, fi] - Log[Abs[AddIndices[sigma, fi]]]]

LogXTransform[e_] :=
e /. LogXTransformRule

AbsPowerTransformRule = Abs[x_]^p_/p_ :>
Block[{sigma = Unique["sigma"], fi = FreeIndices[x]},
         AppendTo[FastNeurons, sigma];
         x AddIndices[sigma, fi] -
Abs[AddIndices[sigma, fi]]^(p/(p-1)) ((p-1)/p)] /; FreeReversedQ[x]
```

```
AbsPowerTransform[e_] :=
e /. AbsPowerTransformRule


(* Index domain notation *)


Parts[IndexDomain[l_, h_]] := 1
Parts[CrossProductDomain[l_, r_]] := Parts[l] * Parts[r]
Parts[DisjointUnion[l_, r_]] := Parts[l] + Parts[r]

Size[IndexDomain[l_, h_]] := 1
Size[CrossProductDomain[l_, r_]] := Size[l] + Size[r]
Size[DisjointUnion[l_, r_]] := If[Size[l] == Size[r], Size[l], 0]

Dimension[n_, IndexDomain[l_, h_]] := If[n == 1, IndexDomain[l, h], 0]
Dimension[n_, CrossProductDomain[l_, r_]] :=
If[n <= Size[l], Dimension[n, l], Dimension[n - Size[l], r]]
Dimension[n_, DisjointUnion[l_, r_]] :=
DisjointUnion[Dimension[n, l], Dimension[n, r]]

Start[part_, size_, IndexDomain[l_, h_]] := If[part == 1 && size == 1, l, 0]
Finish[part_, size_, IndexDomain[l_, h_]] := If[part == 1 && size == 1, h, 0]

Start[part_, size_, CrossProductDomain[l_, r_]] :=
If[size <= Size[l], Start[Floor[(part-1) / Parts[r]] + 1, size, l],
Start[Mod[(part - 1), Parts[r]] + 1, size - Size[l], r]]
Finish[part_, size_, CrossProductDomain[l_, r_]] :=
If[size <= Size[l], Finish[Floor[(part-1) / Parts[r]] + 1, size, l],
Finish[Mod[(part - 1), Parts[r]] + 1, size - Size[l], r]]

Start[part_, size_, DisjointUnion[l_, r_]] :=
If[part <= Parts[l], Start[part, size, l],
    Start[part - Parts[l], size, r]]
Finish[part_, size_, DisjointUnion[l_, r_]] :=
If[part <= Parts[l], Finish[part, size, l],
    Finish[part - Parts[l], size, r]]

DomainQ[id_] := Head[id] == IndexDomain || Head[id] == CrossProductDomain ||
Head[id] == DisjointUnion

SumOver[f_, {i_, IndexDomain[l_, h_]}] := SumOver[f, {i, l, h}]

SumOver[f_, {i_, id_ /; DomainQ[id]}] := Module[{p},
ReplacePart[Table[SumOver[f, {i, Start[p, 1, id], Finish[p, 1, id]}],
{p, Parts[id]}], Plus, 0]]
```

```
SumOver[f_, {i_, j_, id_ /; DomainQ[id]}] := Module[{p},
ReplacePart[Table[SumOver[f, {i, Start[p, 1, id], Finish[p, 1, id]},
     {j, Start[p, 2, id], Finish[p, 2, id]}],
{p, Parts[id]}], Plus, 0]]

SumOver[f_, {i_, j_, k_, id_ /; DomainQ[id]}] := Module[{p},
ReplacePart[Table[SumOver[f, {i, Start[p, 1, id], Finish[p, 1, id]},
     {j, Start[p, 2, id], Finish[p, 2, id]},
     {k, Start[p, 3, id], Finish[p, 3, id]}],
{p, Parts[id]}], Plus, 0]]

SumOver[f_, {i_, j_, k_, l_, id_ /; DomainQ[id]}] := Module[{p},
ReplacePart[Table[SumOver[f, {i, Start[p, 1, id], Finish[p, 1, id]},
     {j, Start[p, 2, id], Finish[p, 2, id]},
     {k, Start[p, 3, id], Finish[p, 3, id]},
     {l, Start[p, 4, id], Finish[p, 4, id]}],
{p, Parts[id]}], Plus, 0]]

TableOf[f_, {i_, IndexDomain[l_, h_]}] := TableOf[f, {i, l, h}]

TableOf[f_, {i_, id_ /; DomainQ[id]}] := Module[{p},
ReplacePart[Table[TableOf[f, {i, Start[p, 1, id], Finish[p, 1, id]}],
{p, Parts[id]}], Plus, 0]]

TableOf[f_, {i_, j_, id_ /; DomainQ[id]}] := Module[{p},
ReplacePart[Table[TableOf[f, {i, Start[p, 1, id], Finish[p, 1, id]},
     {j, Start[p, 2, id], Finish[p, 2, id]}],
{p, Parts[id]}], Plus, 0]]

TableOf[f_, {i_, j_, k_, id_ /; DomainQ[id]}] := Module[{p},
ReplacePart[Table[TableOf[f, {i, Start[p, 1, id], Finish[p, 1, id]},
     {j, Start[p, 2, id], Finish[p, 2, id]},
     {k, Start[p, 3, id], Finish[p, 3, id]}],
{p, Parts[id]}], Plus, 0]]

TableOf[f_, {i_, j_, k_, l_, id_ /; DomainQ[id]}] := Module[{p},
ReplacePart[Table[TableOf[f, {i, Start[p, 1, id], Finish[p, 1, id]},
     {j, Start[p, 2, id], Finish[p, 2, id]},
     {k, Start[p, 3, id], Finish[p, 3, id]},
     {l, Start[p, 4, id], Finish[p, 4, id]}],
{p, Parts[id]}], Plus, 0]]
```

File "Descent.m"

(Descent.m isn't needed in the presence of the back end.)

```
<<Nemesis.m

GradDesc::usage =
    "GradDesc[f] minimizes the objective function f. The optimizer performs
    gradient descent with a fixed step size for 200 steps."

SaddlePoint::usage =
    "SaddlePoint[f, {r1, r2, r3, ...}] finds a saddle point of the objective
    function f by minimizing it with respect to all variables except r1,
    r2, r3, etc. The optimizer performs gradient descent/ascent with a
    fixed step size for 200 steps."

ExtractSymbols[x_] :=
Block[{n = Length[x], i},
    If[n == 0 && Head[x] == Symbol, Return[x], 0];
    If[n == 0 && Head[x] != Symbol, Return[{}], 0];
    If[n == 1, Return[ElimFunc[x]], 0];
    Union[ Flatten[ Table[ ExtractSymbols[ x[[i]] ], {i, n}] ] ]
]

ElimFunc[x_] :=
Block[{},
    If[Head[x] == Abs, Return[{}], 0];
    If[Head[x] == Exp, Return[{}], 0];
    If[Head[x] == Log, Return[{}], 0];
    If[Head[x] == Tanh, Return[{}], 0];
    x
]

max = 1

min = 0

gain = 50

range := 0.5 * (max - min)

avg := 0.5 * (max + min)

Transfer[x_] := gain*x / (1 + Abs[gain*x/range]) + avg

InvTransfer[y_] := (y - avg) / (gain*(1 - Abs[(y - avg)/range]))

Potential[x_] := 10 range*(-range*Log[range - Abs[x-avg]] - Abs[x-avg])/gain
```

```
Abs' := Sign

GradDesc[f_] := InternalGradDesc[f /.          SumOver -> Sum]

GradDesc[f_, nsteps_] := InternalGradDesc[f /.          SumOver -> Sum, nsteps]

InternalGradDesc[f_, nsteps_] :=
Block[{i, j, args = ExtractSymbols[f],
len, table, rules = {}, dxs},
    len = Length[args];
    table = Table[0.1 + 0.2 Random[], {i, len}];
    Print[table];
    For[i=1, i<=len, i++,
dxs[i] = D[f, args[[i]]];
AppendTo[rules, args[[i]] -> table[[i]] ]
    ];
    For[j=0, j<nsteps, j++,
For[i=1, i<=len, i++,
    table[[i]] = table[[i]] - 0.01 (dxs[i] /. rules);
    rules[[i]] = args[[i]] -> table[[i]]
];
Print[j];
Print[table];
    ];
    rules
]

FindReversed[all_, rev_] :=
Flatten[ Map[ Function[ FindName[ #1, rev ] ], all] ]

FindName[x_, names_] :=
If[MatchName[x, names], x, {}]

MatchName[x_, names_] :=
If[MatchQ[Head[x], Symbol],
     MemberQ[names, x], MatchName[Head[x], names]]

SaddlePoint[f_, revnames_] :=
InternalSaddlePoint[f /.          SumOver -> Sum, revnames]

SaddlePoint[f_, revnames_, nsteps_] :=
InternalSaddlePoint[f /.          SumOver -> Sum, revnames, nsteps]

InternalSaddlePoint[f_, revnames_, nsteps_] :=
Block[{i, j, args = ExtractSymbols[f], revargs, len,  revlen,
table, rules = {}, dxs, dxrs},
```

```
    revargs = FindReversed[args, revnames];
    args = Complement[args, revargs];
    len = Length[args];
    revlen = Length[revargs];
    table = Table[0.1 + 0.2 Random[], {i, len + revlen}];
    Print[table];
    For[i=1, i<=len, i++,
dxs[i] = D[f, args[[i]]];
AppendTo[rules, args[[i]] -> table[[i]] ]
    ];
    For[i=1, i<=revlen, i++,
dxrs[i] = D[f, revargs[[i]]];
AppendTo[rules, revargs[[i]] -> table[[i+len]] ]
    ];
    For[j=0, j<nsteps, j++,
For[i=1, i<=len, i++,
    table[[i]] = table[[i]] - 0.01 (dxs[i] /. rules);
    rules[[i]] = args[[i]] -> table[[i]]
];
For[i=1, i<=revlen, i++,
    table[[i+len]] = table[[i+len]] + 0.1 (dxrs[i] /. rules);
    rules[[i+len]] = revargs[[i]] -> table[[i+len]]
]
Print[j];
Print[table];
    ];
    rules
]
```

B Appendix: Algebraic Transformations of Objective Functions

Neural Networks, Vol. 3, pp. 651–669, 1990
Printed in the USA.

0893-6080/90 $3.00 + .00

ORIGINAL CONTRIBUTION

Algebraic Transformations of Objective Functions

ERIC MJOLSNESS AND CHARLES GARRETT

Yale University

(*Received* 27 *March* 1989; *revised and accepted* 31 *January* 1990)

Abstract—*Many neural networks can be derived as optimization dynamics for suitable objective functions. We show that such networks can be designed by repeated* transformations *of one objective into another with the same fixpoints. We exhibit a collection of algebraic transformations which reduce network cost and increase the set of objective functions that are neurally implementable. The transformations include simplification of products of expressions, functions of one or two expressions, and sparse matrix products (all of which may be interpreted as Legendre transformations); also the minimum and maximum of a set of expressions. These transformations introduce new interneurons which force the network to seek a saddle point rather than a minimum. Other transformations allow control of the network dynamics, by reconciling the Lagrangian formalism with the need for fixpoints. We apply the transformations to simplify a number of structured neural networks, beginning with the standard reduction of the winner-take-all network from* $O(N^2)$ *connections to* $O(N)$. *Also susceptible are inexact graph-matching, random dot matching, convolutions and coordinate transformations, and sorting. Simulations show that fixpoint-preserving transformations may be applied repeatedly and elaborately, and the example networks still robustly converge.*

Keywords—Objective function, Structured neural network, Analog circuit, Transformation of objective, Fixpoint-preserving transformation, Lagrangian dynamics, Graph-matching neural net, Winner-take-all neural net.

1. INTRODUCTION

Objective functions have become important in the study of artificial neural networks, for their ability to concisely describe a network and the dynamics of its neurons and connections. For neurons with objective-function dynamics, the now standard procedure (Hopfield, 1984; Hopfield & Tank, 1985; Koch, Marroquin & Yuille, 1986) is to formulate an objective function (called the "objective" in what follows) which expresses the goal of the desired computation, then to derive a local update rule (often a simple application of steepest descent) which will optimize the dynamic variables, in this case the artificial analog neurons. The update rule should ultimately converge to a fixpoint which minimizes the objective and should be interpretable as the dynamics of a circuit.

This procedure is direct but has drawbacks. For example it considers only the goal of the computation and not the cost for attaining the goal or the path taken in doing so. The resulting neural nets can be quite expensive in their number of connections, and for some objectives the associated local update rule has an algebraic form unsuitable for direct implementation as a neural network.

In this paper we show how to modify the standard procedure by interpolating an extra step: after the objective is formulated, it can be algebraically manipulated in a way which preserves its meaning but improves the resulting circuits (e.g., by decreasing some measure of their cost). Then the improved circuit is derived from the modified objective. Often it will be clear from the algebra alone that some savings in number of connections will occur, or that a previously nonimplementable objective has been transformed to a form (e.g., single-neuron potentials plus polynomial interactions (Hopfield, 1984)) whose minimization may be directly implemented as an analog circuit in a given technology. And one interesting class of transformations can establish detailed control over the state-space trajectory followed during optimization.

We do not require that the algebraic manipulation

Acknowledgments: We wish to acknowledge discussions with P. Anandan, Christian Darken, Wojtek Furmansky, Gene Gindi, Drew McDermott, Willard Miranker, John Platt, Petar Simic, and Joachim Utans.

This work was supported in part by AFOSR grant AFOSR-88-0240.

Requests for reprints should be sent to Eric Mjolsness, Yale Computer Science Department, P.O. Box 2158 Yale Station, New Haven, CT 06520.

be done automatically; we just ask whether and how it can be done. Our answer is in the form of a short, nonexhaustive list of very general transformations which can be performed on summands of an objective, provided they are of the requisite algebraic form, without altering the fixpoints of the resulting network. Many of these transformations introduce a relatively small number of new neurons which change the local minima of the original objective into saddle points of the new one, and whose dynamical behavior is to seek saddle points. It also seems likely that many useful and general algebraic transformations await discovery.

Although we use the terminology appropriate to the optimization of objectives for dynamic neurons with fixed connections, much of the theory may apply also to "learning" considered as the optimization of dynamic connections in an unstructured net (e.g., Rumelhart, Hinton & Williams, 1986b) or of some smaller set of parameters which indirectly determine the connections in a structured net (Mjolsness, Sharp & Alpert, 1989b).

In the remainder of this section, we will introduce the ideas by rederiving a well-known network simplification: that of the winner-take-all (WTA) network from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$ connections. Section 2 develops the theory of our algebraic transformations, including the reduction of: squares and products of expressions; a broad class of functions of one and two expressions; the minimum and maximum of a set of expressions; and certain matrix forms which retain their sparseness as they are reduced. Implications for circuit design are discussed, and a major unsolved problem related to the handling of sparse matrices is stated. Further algebraic transformations (section 2.7) allow control of the temporal aspects of the optimization process, by modifying the usual Lagrangian formalism (which uses variational calculus to derive time-reversible dynamics) to accommodate the need for fixpoints in neural network dynamics. All the fixpoint-preserving transformations are cataloged in section 2.8. In section 3, some of the transformations are exercised in design examples. Experimental results are available for a graph-matching network, a random-dot-matching network, and an approximate sorting network which involves a series of fixpoint-preserving transformations. For all these networks, approach to a fixpoint is guaranteed if the minimizing neurons operate at a much slower time scale than the maximizing neurons, but experimentally such convergence is also observed when the two time scales are close; the advantage of the latter mode of operation is that it requires much less time for a network to converge. Finally, a discussion follows in section 4.

1.1. Reversed Linear Neurons in the WTA Network

Consider the ordinary winner-take-all analog neural network. Following (Hopfield & Tank, 1985), such a network can be obtained from the objective

$$E_{\text{wta}}(\mathbf{v}) = \frac{c_1}{2}\left(\sum_i v_i - 1\right)^2 + c_2 \sum_i h_i v_i + \sum_i \phi(v_i) \qquad (c_1 > 0), \quad (1)$$

where (Hopfield, 1984; Grossberg, 1988)

$$\phi(v_i) = \int^{v_i} dx\, g^{-1}(x), \qquad (2)$$

using steepest-descent dynamics

$$\dot{v}_i = -\partial E/\partial v_i \qquad (3)$$

or Hopfield-style dynamics

$$\dot{u}_i = -\partial E/\partial v_i, \quad v_i = g(u_i). \qquad (4)$$

The resulting connection matrix is

$$T_{ij} = -c_1,$$

which implies global connectivity among the neurons: if there are N neurons, there are N^2 connections. It is well known that the winner-take-all circuit requires only $\mathcal{O}(N)$ connections if one introduces a linear neuron σ whose value is always $\Sigma_i v_i$. It is not so well known that this can be done entirely within the objective function, as follows:

$$\hat{E}_{\text{wta}}(\mathbf{v}) = c_1\left(\sum_i v_i - 1\right)\sigma - \frac{c_1}{2}\sigma^2 + c_2 \sum_i h_i v_i + \sum_i \phi(v_i), \quad (5)$$

where the steepest-descent dynamics, for example, is modified to become

$$\begin{aligned} \dot{v}_i &= -r_i \partial \hat{E}/\partial v_i \\ &= -\partial \hat{E}/\partial v_i \quad \text{(specialize to } r_i = 1) \qquad (6) \\ \dot{\sigma} &= +r_\sigma \partial \hat{E}/\partial \sigma \quad (r_\sigma > 0) \\ &= c_1 r_\sigma \left(-\sigma + \sum_i v_i - 1\right) \\ &= 0 \text{ if } \sigma = \sum_i v_i - 1. \qquad (7) \end{aligned}$$

But if $\sigma = \Sigma_i v_i - 1$ then one can calculate that $\partial E/\partial v_i = \partial \hat{E}/\partial v_i$; thus eqns (1) and (5) have the same fixpoints. The connectivity implied by counting the monomials in eqn (5) is $\mathcal{O}(N)$ connections, the minimum possible for this problem.

Note that the σ linear neuron actually behaves so as to *increase* the objective $E(V, \sigma)$, while the v_i neurons act to decrease it; σ may be called a *reversed* neuron. Reversed neurons introduce a new element

of competition into a network; indeed, two-person zero-sum games are usually modelled using objectives which one player increases and the other decreases (von Neumann and Morgenstern, 1953). So in this network, and in the others we will introduce, minimization is replaced with finding a saddle point and the problem becomes hyperbolic. This follows immediately from the sign of σ^2 in (5), which eliminates all local minima. Fortunately there are hyperbolic versions of such efficient optimization procedures as the conjugate gradient method; Leunberger (Luenberger, 1984) gives two examples.

For finite r_σ, σ is a delayed version of the sum $\Sigma_i v_i - 1$ and although the network dynamics are different from eqn (3), the fixed point is the same. Alternatively, the rate parameter r_σ may be adjusted to make σ move at a different time scale from the rest of the neurons. As r_σ approaches infinity, σ becomes an *infinitely fast* neuron whose value is always

$$\sigma = \sum_i v_i - 1$$

and the other neurons see an effective objective

$$\hat{E}_{wta}(\mathbf{v}, \sigma(\mathbf{v})) = E_{wta}(\mathbf{v}) \tag{8}$$

so that their dynamics become identical to that of the original fully connected winner-take-all network, which is guaranteed to approach a fixpoint since $dE_{wta}/dt < 0$ and E_{wta} is bounded below. There are very efficient serial and parallel implementations for networks with $r_\sigma \to \infty$, which update the infinitely fast neuron whenever its ordinary neighbors change; this is a standard trick in neural network and Monte Carlo physics simulations. When it is applied to simplify the simulation of the WTA equations of motion, we arrive at the standard WTA trick.

Moody (Moody, 1989) independently discovered an objective function equivalent to (5) and first simulated its delayed Hopfield-style equations of motion (see (4)). But he remained unaware that the σ neuron acts to maximize rather than minimize the objective, driving the system to a saddle point. Platt and Barr (1988, 1987) performed constrained optimization (including multiple WTA constraints) by using a subset of neurons that explicitly increase the objective, and hence a network that seeks saddle points rather than local minima. Indeed reversed neurons are a generalization of their analog Lagrange multiplier neurons, which were found earlier in a non-neural context by Arrow (1958).

Both reversed neurons and Lagrange multiplier neurons act to maximize an objective which other neurons act to minimize. The difference is that Lagrange multiplier neurons must appear linearly in the objective. General reversed neurons can have self-interactions; in particular, the potential of a linear reversed neuron like σ in the winner-take-all network is

$$-\frac{1}{2g_0}\sigma^2 = \int^{\sigma} dx\, g^{-1}(x), \qquad \text{where } g(x) = -g_0 x, \tag{9}$$

that is, linear reversed neurons have linear transfer functions with negative gain. Thus in circuit language they are just inverters, which happen to occur in a network with an objective function and to act so as to increase E. Lagrange multiplier neurons, on the other hand, have no potential term and are not inverters. It is also worth noting that Lagrange multiplier neurons work best in conjunction with an additional penalty term $h(\vec{v})^2/2$ (where $h(\vec{v}) = 0$ is the constraint) and the penalty term can be efficiently implemented using one reversed neuron per constraint, as we will see.

If in addition to being reversed, a neuron is also infinitely fast, then it may be necessary to restrict its connectivity in order to efficiently simulate or implement the network. One possible design rule is to entirely prohibit connections between infinitely fast neurons; this prevents one from having to solve a system of linear equations in order to update a set of infinitely fast neurons. We will *not* generally assume that reversed neurons are infinitely fast.

2. THEORY

The reversed linear neuron is applicable in many circumstances beyond the winner-take-all network. We can begin to see its generality by considering objectives of the form

$$E(\mathbf{v}) = E_0(\mathbf{v}) + (c/2)X^2(\mathbf{v}),$$

where E_0 and X are any algebraic expressions, and c is a constant of either sign. This may be transformed to

$$\hat{E}(\mathbf{v}, \sigma) = E_0 + cX\sigma - (c/2)\sigma^2$$

and if X is a polynomial, this represents a reduction in the number and order of the monomials that occur in E. The transformation technique used here is simple to state: find squared expressions $(c/2)X^2$ as summands in the objective function, and replace them with $cX\sigma - (c/2)\sigma^2$. Here c must be a constant and σ is a new linear interneuron, reversed if c is positive. Most, but not all, of our reversed linear interneurons will be introduced this way. The transformation may be abbreviated as

$$\tfrac{1}{2}X^2 \longrightarrow X\sigma - \tfrac{1}{2}\sigma^2. \tag{10}$$

We employ the steepest-ascent-descent dynamics

$$\begin{aligned}\dot{v}_i &= -\partial \hat{E}/\partial v_i \\ &= -\partial E_0/\partial v_i - c\sigma \partial X/\partial v_i, \\ \dot{\sigma} &= +(r_\sigma/c)\partial \hat{E}/\partial \sigma \\ &= r_\sigma(X - \sigma),\end{aligned}$$

which, at a fixpoint, has $X = \sigma$ and $\partial E_0/\partial v_i + cX\partial X/\partial v_i = \partial E/\partial v_i = 0$. Likewise a fixpoint of E can be extended, by setting $\sigma = X$, to a fixpoint of $\hat{E}$. So fixpoints are preserved by the transformation (10). The argument also works if some of the v_i are already reversed neurons, so that $\dot{v}_i = \pm\partial \hat{E}/\partial v_i$.

As an example of the transformation (10), one can robustly implement a constraint $h(\vec{v}) = 0$ using both a penalty term $ch^2/2$ and a Lagrange multiplier neuron λ. The objective becomes

$$E_{\text{constraint}}(\mathbf{v}, \sigma, \lambda) = c\sigma h(\mathbf{v}) - c\sigma^2/2 + \lambda h(\mathbf{v}). \quad (11)$$

2.1. Products and Order Reduction

From the transformation (10) we may deduce two others, which are applicable whenever a summand of an objective is a product XY of expressions X and Y. Such products are common and can be expensive; for example if X and Y are each sums of N variables, then expanding their product out into monomial interactions gives N^2 connections. But only $\mathcal{O}(N)$ connections are needed if one uses the transformation

$$\begin{aligned}XY &= \tfrac{1}{2}(X + Y)^2 - \tfrac{1}{2}X^2 - \tfrac{1}{2}Y^2 \\ &\longrightarrow (X + Y)\sigma - X\tau - Y\omega \\ &\quad - \tfrac{1}{2}\sigma^2 + \tfrac{1}{2}\tau^2 + \tfrac{1}{2}\omega^2 \\ &\longrightarrow X(\sigma - \tau) + Y(\sigma - \omega) \\ &\quad - \tfrac{1}{2}\sigma^2 + \tfrac{1}{2}\tau^2 + \tfrac{1}{2}\omega^2. \end{aligned} \quad (12)$$

Here σ is a reversed linear neuron, and τ and ω are ordinary linear neurons. If all three linear interneurons are infinitely fast, which is easy to stimulate since they are not directly connected, then the transformation does not change the dynamics of the rest of the variables in the network. Otherwise, the dynamics and the basins of attraction change, but the network fixed points remain the same.

This transformation may be simplified,[1] to

$$\begin{aligned}XY &= \tfrac{1}{4}[(X + Y)^2 - (X - Y)^2] \\ &\longrightarrow \tfrac{1}{2}(X + Y)\sigma - \tfrac{1}{2}(X - Y)\tau - \tfrac{1}{4}\sigma^2 + \tfrac{1}{4}\tau^2 \\ &= \tfrac{1}{2}X(\sigma - \tau) + \tfrac{1}{2}Y(\sigma + \tau) - \tfrac{1}{4}\sigma^2 + \tfrac{1}{4}\tau^2. \end{aligned} \quad (13)$$

Compared to eqn (12), this transformation results in the same number of monomial interactions and one less neuron, which may be useful on occasion.

Reversed neurons allow one to transform a high-order polynomial objective, monomial by monomial, into a third-order objective. Similar transformations on the neural networks or analog circuits are well known. But it is easier to do theoretical work with the objective, and by transforming the objective first, and then translating to a neural net, one can obtain novel third-order neural nets.

We may expand a high-order polynomial objective into monomials, each of which corresponds to one connection or "synapse" of the associated neural network. We may reduce the order of an entire objective by reducing the order of each monomial. Consider, then, a single fourth-order monomial:

$$E_{\text{mono}}(x, y, z, w) = -Txyzw, \quad (14)$$

which by (12) may be transformed to

$$\begin{aligned}\hat{E}_{\text{mono}}(x, y, z, w, \sigma, \tau, \omega) &= T[xy(\tau - \sigma) \\ &\quad + zw(\omega - \sigma) + \tfrac{1}{2}\sigma^2 - \tfrac{1}{2}\tau^2 - \tfrac{1}{2}\omega^2]. \end{aligned} \quad (15)$$

Here σ, τ, and ω are linear neurons with gain $1/T$. The order-reducing transformation is illustrated in network language in Figure 1.

The same technique may be used to recursively transform a monomial of any order m to a sum of third-order monomials, plus potentials for the new linear interneurons. The resulting number of new monomials and interneurons is $<\mathcal{O}(m(\log m)^2)$ if the reduction is done in a balanced way and if expressions like $X(\sigma - \tau)$ are not expanded out to $X\sigma - X\tau$ during the reduction.

Another order-reduction transformation, superior in some circumstances, will be developed in section 2.4.

2.2. Reducing $F(X)$

Often an objective function includes a fairly general nonlinear function F of an entire expression X. This

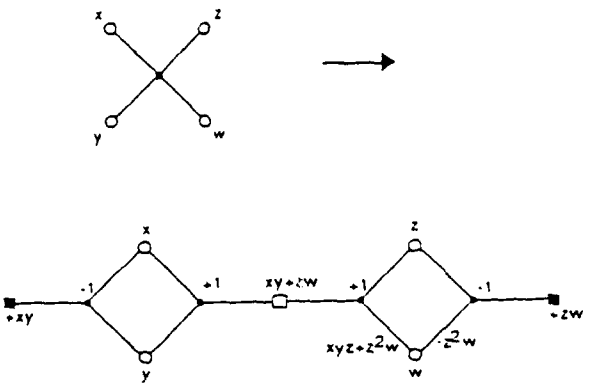


FIGURE 1. Order reduction. Arbitrary fourth- to third-order reduction, using linear interneurons. Open circles: original neurons. Open squares: ordinary linear interneurons. Closed squares: reversed linear interneurons. Dots: connections, with strengths as indicated. Equilibrium values of the interneurons are indicated. After the transformation, neuron w receives input $xyz + z^2w$ from the left and a compensating input of $-z^2w$ from the right.

[1] As pointed out to us by P. Anandan.

may be much more difficult and costly to implement directly than either a low-order polynomial or a single-neuron potential function $\phi_i(v_i)$, because the $F(X)$ nonlinearity involves an interaction of many variables. But for some functions F, such an algebraic form is still neurally implementable by means of transformations:

$$\int^X f(u)\,du \longrightarrow X\sigma - \int^\sigma f^{-1}(u)\,du \quad (f \text{ invertable}). \tag{16}$$

Note that X no longer appears inside the function $F = \int f$. The validity of this transformation (eqn (16)) may be proven by noting that the optimal value of σ is $f(X)$, and then either integrating by parts the expression

$$\int^{f(X)} f^{-1}(u)\,du = \int^X vf'(v)\,dv,$$

or else differentiating both pre- and post-transformation objectives with respect to X.

In this way one can treat functions $\exp X$, $|X|\log|X|$, $\log|X|$, and $|X|^p$ of arbitrary algebraic expressions X:

$$\begin{aligned} e^X &\longrightarrow (X + 1)\sigma - \sigma\log\sigma, \\ |X|\log|X| &\longrightarrow |X|(\sigma + 1) - e^\sigma, \\ \log|X| &\longrightarrow X\sigma - \log|\sigma|, \\ \frac{1}{1+p}|X|^{1+p} &\longrightarrow X\sigma - \frac{1}{1+1/p}|\sigma|^{1+1/p} \\ &\quad (p \neq -1, 0). \end{aligned} \tag{17}$$

Thus, neural nets may be constructed from some highly nonpolynomial objectives.

The interneurons may still be reversed, but are no longer linear, in this kind of transformation. The potential $\phi(\sigma)$ permits a possible efficiency technique. The dynamics of eqn (4) is expected to be more efficient than eqn (3), since it may be viewed as a quasi-Newton method which takes into account the potential but not the interaction part of a neural net objective (as shown by J. Utans, 1989). A related update scheme for the σ reversed interneuron is

$$\begin{aligned} \sigma_i &= f(s_i), \quad & \dot{s}_i &= r_\sigma\,\partial E/\partial\sigma_i = r_\sigma(X_i - s_i), \\ v_i &= g_i(u_i), \quad & \dot{u}_i &= -\partial E(\mathbf{v}, \sigma)/\partial v_i, \end{aligned} \tag{18}$$

which is an alternative to direct steepest-ascent-descent. This dynamics has the distinct advantage of a simple interpretation in terms of analog electrical circuits (Hopfield, 1984). For example, $F(X) = X(\log X - 1)$ requires a special neuron whose transfer function is logarithmic. This can be provided, approximately and within a mildly restricted domain of the input values, in analog VLSI (Sivilotti, Mahowald & Mead, 1987). Similarly $F(X) = \log X$ would require a transfer function $f(s) = 1/s$, and an exponential transfer function would lead to $F(X) = \exp X$. It might be possible to characterize a particular technology by a list of the basic forms of objectives it makes available, with their respective costs and restrictions, and to compile general networks into the desired forms by using a catalog of algebraic transformations.

Equation (16) generally may be used to transform a term $F(X)$ in an objective by transferring the nonlinearity due to F from $F(X)$ to the single-neuron potential $\phi(\sigma) = \int^\sigma f^{-1}(u)\,du$. By transferring the nonlinearity from an interaction term (i.e., a summand of the objective which involves several dynamical variables) to a single-neuron potential term, one cannot only decrease the cost of implementing a network which uses gradient methods for optimization, but one can transform unimplementable objectives into implementable ones. For example, one might regard the class of multivariable polynomials as the "implementable" interactions in a certain technology (Rumelhart, Hinton & McClelland, 1986a). (In this case the word "interaction" is usually reserved for a multivariable monomial, out of which polynomials are built by addition of objectives.) Then one might use eqn (16) to reduce other, far more general interaction objectives to the implementable form.

Of course the required potential $\phi(\sigma)$ may itself be "unimplementable", but approximating its gradient with a small circuit is likely to be far more tractable than approximating $\nabla F(X)$ because ϕ, unlike F, is a function of just one dynamic variable. An approximation of $\phi'(\sigma)$ might be formulated as

$$\phi(\sigma) \longrightarrow \hat{\phi}(\sigma) = \sum_\alpha c_\alpha \hat{\phi}_\alpha(\sigma),$$

where each $\hat{\phi}(\sigma)$ is regarded as an implementable self-interaction and c_α are adjustable coefficients.

2.3. Reducing $F(X)$: Examples

We exhibit two examples of the objective function transformation of eqn (16), with dynamics (18). First consider the toy problem of optimizing

$$E(x) = e^x + e^{-x}.$$

One of the exponentials may be taken to be the potential of a neuron whose transfer function is exponential; such a transfer function is implementable in many technologies and, like the logarithmic transfer function, might be part of a standard component library for analog neural nets. Adding the other exponential would however require a special modification to this transfer function, and their sum might not be in the standard component library. So let us move the second exponential nonlinearity to another

neuron whose transfer function is logarithmic:

$$\dot{E}(x, \sigma) = e^x - x\sigma - \sigma \log \sigma + \sigma,$$

whence the dynamics (18) become

$$\begin{aligned} \sigma &= e^x, \qquad \dot{s} = r_\sigma(-x - s), \\ s &= \log u, \quad \dot{u}_i = r_x(\sigma - u). \end{aligned} \tag{19}$$

The evolution of this two-neuron network is shown in Figure 2, along with a contour map of the saddle-shaped objective $\dot{E}$. Note that for quick descent, $r_x \approx r_\sigma$ is preferred. Despite the saddle point, and despite the potential numerical sensitivity of exponential and logarithmic transfer functions, the network functions well.

A second example is the linear programming network of (Tank & Hopfield, 1986) which can also be interpreted as an application of transformation (16) with $r_\sigma \to \infty$ in the dynamics of eqn (18). The linear programming problem is to minimize $\mathbf{A} \cdot \mathbf{v}$ subject to a set of constraints $\mathbf{D}_j \cdot \mathbf{v} \geq B_j$. Their objective is

$$E[\mathbf{v}] = \sum_i A_i v_i + \sum_j F\left(\sum_i D_{ji} v_i - B_j\right) + \sum_i v_i^2/g_0 \quad (\text{large } g_0),$$

where $dF(x)/dx = f(x) - \max(0, -x)$ penalizes violations of the inequality constraints and proved to be electronically implementable. Transforming according to (16) we get

$$\dot{E}[\mathbf{v}] = \sum_i A_i v_i + \sum_j \sigma_j \left(\sum_i D_{ji} v_i - B_j\right) - \sum_j \int^{\sigma_j} f^{-1}(s)\, ds + \sum_i v_i^2/g_0$$

and equation of motion

$$\begin{aligned} \sigma_j &= f(s_j), \quad \dot{s}_j = r_\sigma\left(\sum_i D_{ji} v_i - B_j - s_j\right) \\ v_i &= g_0 u_i, \quad \dot{u}_i = r_v\left(-u_i - A_i - \sum_j D_{ji}\sigma_j\right). \end{aligned} \tag{20}$$

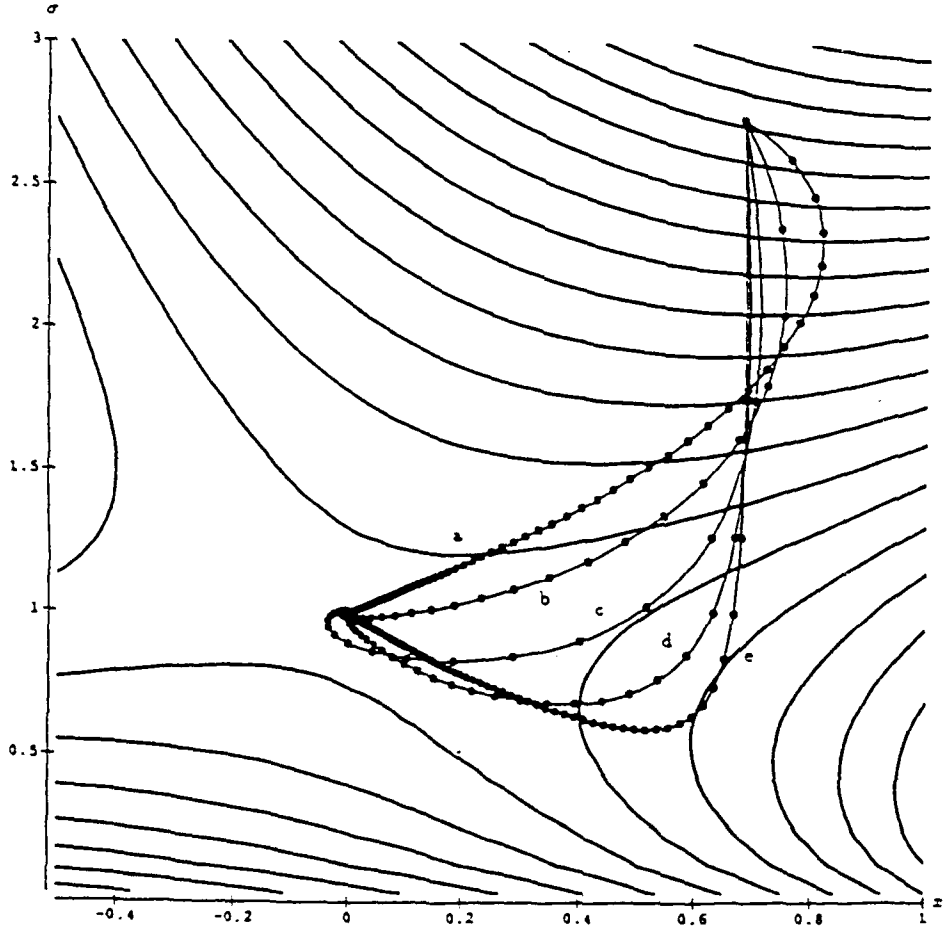


FIGURE 2. Exponential and logarithmic neurons. The minimum of $e^x + e^{-x}$ occurs at the saddle point of $e^x - x\sigma - \sigma \log \sigma + \sigma$, whose contours are plotted here. Also various two-neuron trajectories to the saddle point are shown, in which x or σ moves more slowly than the fastest implementable time scale, assumed to be $r = 1$. (a) $r_x = 1$, $r_\sigma = .1$, (b) $r_x = 1$, $r_\sigma = .3$, (c) $r_x = 1$, $r_\sigma = 1$, (d) $r_x = .3$, $r_\sigma = 1$, (e) $r_x = .1$, $r_\sigma = 1$. Dots occur every 10 time constants, so (c) gives quickest convergence.

This network approaches a saddle point rather than a minimum, but the $r_a \to \infty$ version,

$$\dot{u}_i/r_a = -u_i - A_i - \sum_j D_{ij} f\left(\sum_i D_{ij} v_i - B_j\right)$$

is exactly the network dynamics of eqn (17) of (Tank & Hopfield, 1986).

2.4. Interacting Expressions: Reducing $G(X, Y)$

Until now we have attempted to reduce all interactions to the forms xy and xyz, but those may not be the only cost-effective few-variable interactions allowed by a given physical technology. If others are allowed, then there are one-step transformations for a class of functions of two arbitrary expressions $G(X, Y)$. In this way the set of objectives with known low-cost neural implementations can be expanded to include common algebraic forms such as X/Y and $XF(Y)$.

From eqn (16) we may derive a generalization to functions of two expressions, $G(X, Y)$:

$$\int^X du \left[\int^Y dv\, g(u, v)\right]^{-1} (u) \text{ [note function inverse]}$$
$$\longrightarrow X\sigma - \int^\sigma du \int^Y dv\, g(u, v) \text{ [by (16)]}$$
$$= X\sigma - \int^Y dv \int^\sigma du\, g(u, v)$$
$$\longrightarrow X\sigma - Y\tau + \int^\tau dv \left[\int^\sigma du\, g(u, v)\right]^{-1} (v)$$
[by (16)].

Thus,

$$\int^X du \left[\int^Y dv\, g(u, v)\right]^{-1} (u) \longrightarrow X\sigma - Y\tau + \int^\tau dv \left[\int^\sigma du\, g(u, v)\right]^{-1} (v). \quad (21)$$

Of course the inverse functions must exist for this transformation to be valid, and this restricts G.

Taking

$$g(u, v) = 1/2\sqrt{uv}$$

we can derive the transformation $-Y/X \to X\sigma - Y\tau - \sigma/\tau$. Rescaling Y and σ by -1, then switching X for Y and σ for τ, this is equivalent to

$$X/Y \longrightarrow X\sigma - Y\tau + \tau/\sigma \quad (22)$$

which is linear in τ but effectively nonlinear due to the optimization of σ.

From eqn (21), Appendix A derives two transformations for the special form $YF(X)$:

$$YF(X) \longrightarrow -X\sigma + Y\tau + \sigma F^{-1}(\tau) \quad (23)$$

(which implies (22)) and

$$Y \int^X f(u)\, du \longrightarrow XY\sigma - Y \int^\sigma du\, f^{-1}(u) \quad (24)$$

assuming $f = F'$ is invertable and $f^{-1} = (F')^{-1}$ is differentiable.

Monomial order reduction can sometimes be accomplished more cheaply using $x \log y$ interactions than third-order ones. If v_i are all restricted to be positive, then

$$\prod_{i=1}^m v_i = \exp \sum \log v_i$$
$$\longrightarrow \sigma\left(\sum_i \log v_i + 1\right) - \sigma \log \sigma \quad (25)$$
$$= \sum_i \sigma \log v_i + \sigma(1 - \log \sigma).$$

This objective has $\mathcal{O}(m)$ interactions of the new type. The fixpoint value of σ is $\Pi_i v_i$ at which point the steepest-descent input to v_i is $-\sigma/v_i = -\Pi_{j \neq i} v_j$.

A product of expressions could be further reduced using eqn (23) and xe^y interactions:

$$\prod_{\alpha=1}^m |X_\alpha| \longrightarrow \sum_\alpha (\sigma\tau_\alpha - |X_\alpha|\omega_\alpha + \omega_\alpha e^{\tau_\alpha}) + \sigma(1 - \log \sigma). \quad (26)$$

It has been pointed out to us (Simic, 1989) that the transformations for $F(X)$ and $G(X, Y)$, and hence all the transformations discussed so far, can be interpreted as Legendre transformations (Courant & Hilbert, 1962).

2.5. Min and Max

The minimum or maximum of a set of expressions $\{X_\alpha\}$ can be implemented using a winner-take-all network in which each expression is represented by a neuron σ_α which competes with the others and is constrained to lie between 0 and 1. Indeed, $\Sigma_\alpha X_\alpha \sigma_\alpha$ attains the value $\min_\alpha X_\alpha$ when the correct representative wins, and also provides inhibition to the representatives in proportion to the values of the expressions they represent, so that the correct representative will win. The potential $\phi(\sigma)$ that occurs in the WTA network must have only one minimum, so that there is no hysteresis in the circuit, and must closely approximate a square well (i.e., must have high gain). Under these circumstance, we can transform

$$E = \min_\alpha X_\alpha \longrightarrow \sum_\alpha X_\alpha \sigma_\alpha + C\left(\sum_\alpha \sigma_\alpha - 1\right)\lambda - \frac{C}{2}\lambda^2 + \sum_\alpha \phi(\sigma_\alpha) \quad (27)$$

(where C and the gain of ϕ are sufficiently large) and at any fixpoint of $\bar{\sigma}$ the derivatives of E with respect to all other dynamical variables will be preserved. So, fixpoints will be preserved.

Likewise

$$\max_a X_a \longrightarrow \sum_a X_a\sigma_a - C\left(\sum_a \sigma_a - 1\right)\lambda + \frac{C}{2}\lambda^2 - \sum_a \phi(\sigma_a). \quad (28)$$

An alternate transformation for max proceeds through the identity

$$\max_a X_a = \lim_{p\to\infty}\left(\sum_a X_a^p\right)^{1/p} \quad (X_a > 0). \quad (29)$$

These transformations have application in the design of some standard "content addressable memories" for which the ideal objective, which must be translated to a form polynomial in its interactions, may be taken as

$$E_{CAM}(\mathbf{v}) = -\max_{\text{memories } m} \mathbf{v}\cdot\mathbf{v}^m - \varepsilon\mathbf{v}\cdot\mathbf{h}_{input} + \sum_i \phi_{=1}(v_i)$$

with $-1 \le v_i \le 1$. Using the transformation (28) yields a CAM with one "grandmother neuron" (representative neuron) per memory, and a WTA net among them (closely related to a network described in Moody (1989)):

$$\dot{E}_{CAM}(\mathbf{v}, \sigma) = -\sum_{m,i} v_i^m v_i \sigma_m - \varepsilon\mathbf{v}\cdot\mathbf{h}_{input} + C\left(\sum_m \sigma_m - 1\right)\lambda - C\lambda^2/2 + \sum_m \phi_{\ge 1}(\sigma_m) + \sum_i \phi_{=1}(v_i).$$

Another efficient CAM design (Gindi, Gmitro & Parthasarathy, 1987; Moody, 1989) may be derived by applying transformation (29) to the max expression:

$$\max_m |\mathbf{v}\cdot\mathbf{v}^m| = \left(\sum_m |\mathbf{v}\cdot\mathbf{v}^m|^p\right)^{1/p} \quad (p \text{ large}),$$

which we replace by a monotonic function thereof, $(1/p)\,\Sigma_m |\mathbf{v}\cdot\mathbf{v}^m|^p$, possibly adjusting ε and ϕ to compensate. (At this point one could take $p = 2$ to get a quadratic expression, and regenerate the content addressable memory of Hopfield, 1984.) Then using (17d) we find an implementable neural net objective for large p:

$$\dot{E}_{CAM}(\mathbf{v}, \sigma) = -\sum_{m,i} v_i^m v_i \sigma_m - \varepsilon\mathbf{v}\cdot\mathbf{h}_{input} + \frac{(p-1)}{p}\sum_m |\sigma_m|^{p/(p-1)} + \sum_i \phi_{=1}(v_i).$$

2.6. Matrices, Graphs, and Pointers

One can apply order reduction to objectives containing polynomials of matrices. For dense $N \times N$ matrices, a typical term like $\mathrm{tr}\,\prod_{i=1}^{L} A^{(i)}$ contains N^L scalar monomial interactions, but this can be reduced to $\mathcal{O}(N^3\,L(\log L)^2)$. (Here tr $A \equiv$ trace of $A \equiv \Sigma_i A_{ii}$.) To show this we need only establish the matrix analog of transformation (12), which upon iteration can reduce an Lth order matrix monomial to $\mathcal{O}(L(\log L)^2)$ *third-order matrix monomials like* ABC. Each of these involves N^3 scalar monomial interactions.

Using eqn (12) (one could also use (13)), one can reduce tr XY, where X and Y are now matrix-valued expressions. This form has exactly the same generality as tr XY^T (where $Y^T \equiv$ transpose of Y).

$$\begin{aligned}\mathrm{tr}\,XY^T &= \sum_{ij} X_{ij}Y_{ij} \\ &\longrightarrow \sum_{ij}\left(X_{ij}(\sigma_{ij} - \tau_{ij}) + Y_{ij}(\sigma_{ij} - \omega_{ij}) - \tfrac{1}{2}\sigma_{ij}^2 + \tfrac{1}{2}\tau_{ij}^2 + \tfrac{1}{2}\omega_{ij}^2\right) \\ &= \mathrm{tr}\,X(\sigma - \tau)^T + \mathrm{tr}\,Y(\sigma - \omega)^T - \tfrac{1}{2}\mathrm{tr}\,\sigma\sigma^T + \tfrac{1}{2}\mathrm{tr}\,\tau\tau^T + \tfrac{1}{2}\mathrm{tr}\,\omega\omega^T.\end{aligned} \quad (30)$$

This transformation preserves the sparseness of X and Y in the following sense: if $X_{ij} = Y_{ij} = 0$ at a fixed point, then $\sigma_{ij} = \tau_{ij} = \omega_{ij} = 0$ and the contribution of these neurons to the gradient of the objective is also zero.

A major problem in neural network research (c.f. Feldman, 1982) is to reduce the cost of networks which manipulate graphs. Usually (Hopfield & Tank, 1985, 1986; Mjolsness, Gindi & Anandan, 1989a) objectives for such problems involve dense matrices of neurons representing all the possible links in a graph. But the graphs that arise in computer science and in computer programming usually have a relatively small number of links per node, and are therefore representable by sparse matrices. (If a sparse matrix's entries are all zero or one, it is equivalent to a set of "pointers" in many current computer languages. Pointers are used ubiquitously, wherever some fluidity in data representation is required.) Since we have just shown how to reduce a wide class of matrix objective functions to summands of the form tr ABC while retaining sparseness, it becomes important to reduce this form further by exploiting the sparseness of the matrices involved:

$$\mathrm{tr}\,ABC \longrightarrow ?$$

where A, B, and C are sparse-matrix-valued dynamical variables. We do not yet know how to do this correctly.

One approach to this problem is through the use of codes, like the binary code, which can concisely name the pair of nodes connected by each nonzero matrix element. Zero matrix elements are not ex-

plicitly encoded and this is the advantage of the method. A disadvantage of such codes, for objective functions, is that the configuration space is altered in such a way that new local minima may be introduced, though the old ones will be preserved. A code which allows order reduction to proceed most advantageously is used in the sorting networks of section 3.4.

Another approach is used by Fox and Furmansky (1988). Their load-balancing network involves binary encoding, but the network evolution is divided into a number of phases in which different classes of neurons are allowed to vary while most neurons are held constant. The connections are different from one phase to the next, and do not recur, so that the network is not directly implemented in terms of a circuit but rather requires "virtual" neurons and connections. A virtual neural network can be provided by suitable software on general-purpose computers, or perhaps by further objective function transformations leading to a real (and efficient) neural circuit; the latter alternative has not be achieved.

2.7. Control of Neural Dynamics

So far we have exclusively considered steepest-ascent-descent dynamics such as eqns (3), (4), or (18), which allow little control over the temporal behavior of a network. Often one must design a network with nontrivial temporal behaviors such as running longer in exchange for less circuitry, or focussing attention on one part of a problem at a time. We discuss two algebraic transformations which can be used to introduce detailed control of the dynamics with which an objective is extremized.

One transformation, developed by one of the authors in collaboration with W. Miranker (Mjolsness & Miranker, 1990), replaces an objective E with an associated Lagrangian functional to be extremized in a novel way:

$$E[\mathbf{v}] \longrightarrow L[\dot{\mathbf{v}}|\mathbf{q}] = \int dt \left(K[\dot{\mathbf{v}}, \mathbf{v}|\mathbf{q}] + \frac{dE}{dt} \right),$$
$$\delta L / \delta \dot{v}_i(t) = 0. \quad (31)$$

Here $\mathbf{q}$ is a set of control parameters, and K is a cost-of-movement term independent of the problem and of E. The integrand $\mathfrak{L} = K + dE/dt$ is called the Lagrangian density. Ordinarily in physics, Lagrangian dynamics have a conserved total energy which prohibits convergence to fixed points. Here the main difference is the unusual functional derivative with respect to $\dot{v}$ rather than v. This is a "greedy" functional derivative, in which the trajectory is optimized from beginning to end by repeatedly choosing an extremal value of $v(t)$ without considering its effect on any subsequent portion of the path:

$$\frac{\delta}{\delta v_i(t)} \int_{-\infty}^{t} dt' \, \mathfrak{L}[\dot{\mathbf{v}}, \mathbf{v}] \approx \delta(0) \frac{\partial \mathfrak{L}[\dot{\mathbf{v}}, \mathbf{v}]}{\partial \dot{v}_i(t)}$$
$$= \delta(0) \frac{\partial}{\partial \dot{v}_i(t)} \int_{-\infty}^{t} dt' \, \mathfrak{L}[\dot{\mathbf{v}}, \mathbf{v}] \propto \frac{\partial L}{\partial \dot{v}_i(t)}. \quad (32)$$

Since

$$\frac{\delta L}{\delta \dot{v}_i} = \frac{\partial K}{\partial \dot{v}_i} + \frac{\partial E}{\partial v_i}, \quad (33)$$

transformation (31) preserves fixpoints if $\partial K / \partial \dot{\mathbf{v}}_i) = 0 \Leftrightarrow \dot{\mathbf{v}} = 0$.

For example, with a suitable K one may recover and improve upon steepest-ascent-descent dynamics:

$$E[\mathbf{v}] \longrightarrow L[\dot{\mathbf{v}}|r, \vec{s}] = \int dt \left(\sum_i s_i \phi_{\pm 1}(\dot{v}_i / r) + \sum_i (\partial E / \partial v_i) \dot{v}_i \right), \quad (34)$$
$$0 = \delta L / \delta \dot{v}_i(t) = s_i \phi'_{\pm 1}(\dot{v}_i / r)/r + dE/dv_i, \text{ i.e.}$$
$$\dot{v}_i = r g_{\pm 1}(-s_i r \, dE/dv_i),$$

where the transfer function $-1 \le g_{\pm 1}(x) \le 1$ reflects a velocity constraint $-r \le \dot{v}_i \le r$, and as usual $g = (\phi')^{-1}$. The constants $s_i = 1/s_i = \pm 1$ are used to determine whether a neuron attempts to minimize or maximize E and L. If all $s_i = 1$ then $dE/dt \le 0$ and eqn (34) is a descent dynamics.

Another transformation (proposed and subjected to preliminary experiments in Mjolsness, 1987) can be used to construct a new objective for the control parameters, q, through their effect on the trajectory $v(t)$:

$$E[\mathbf{v}] \longrightarrow \hat{E}[\mathbf{q}] = \sum_i \left(\frac{\partial E}{\partial v_i} \right) s_i \dot{v}_i + \hat{E}_{\text{cost}}[\mathbf{q}]$$
$$= \frac{dE}{dt} + \hat{E}_{\text{cost}}[\mathbf{q}], \quad \text{if all } s_i = 1. \quad (35)$$

In (Mjolsness, 1987) the $s_i = 1$ version of transformation (35) (but not (34)) was used to introduce a computational "attention mechanism" for neural nets as follows. Suppose we can only afford to simulate R out of $N \gg R$ neurons at a time in a large net. The R neurons can be chosen dynamically via control parameters $q_i = r_i \in [0, 1]$ in eqn (34), with $r_i \approx 0$ for all but R active neurons. For high-gain $g_{\pm 1}$ we have $g_{\pm 1}(x) \approx \text{sgn}(x)$ and

$$\frac{dE}{dt} = \sum_i \frac{dE}{dv_i} \dot{v}_i$$
$$= \sum_i r_i \frac{dE}{dv_i} g_{\pm 1}\left(-r_i \frac{dE}{dv_i} \right) \approx -\sum_i r_i \left| \frac{dE}{dv_i} \right|.$$

(Whether the gain is high or not, $g_{\pm 1}$ is an odd function so $dE/dt \leq 0$ and any dynamics for r yields a descent algorithm for E.) The objective for $\mathbf{r}$ is

$$\hat{E}[\mathbf{r}] = -\sum_i r_i \left|\frac{dE}{dv_i}\right| + \frac{A}{2}\left(\sum_i r_i - R\right)^2 + \sum_i \phi_{0/1}(r_i)$$

which describes an R-winner version of a WTA network that determines which R neurons should be active and which $N - R$ should be frozen.

An especially simple and cheap dynamical system for r is to keep all r_i constant most of the time, but every so often to interrupt the simulation of $\delta L/\delta \dot{\mathbf{v}} = 0$ and completely relax $\hat{E}[\mathbf{r}]$. This amounts to re-sorting the neurons v_i according to their gradient magnitudes $|\partial E/\partial v_i|$, and selecting only the first R neurons to be active ($r_i \approx 1$). When simulating a sparsely connected neural net on other underlying hardware, this algorithm can be implemented very cheaply since most v gradients are unchanged between phases of r relaxation, and therefore the new sorted order is just a minor refinement of the previous one. The result is necessarily a descent algorithm for $E[\mathbf{v}]$, with only R neurons active at any time.

2.8. List of Transformations

Let X and Y be any algebraic expressions, containing any number of variables. We list the following fixpoint-preserving transformations of objectives, or summands thereof:

1.1	$\frac{1}{2}X^2 \longrightarrow X\sigma - \frac{1}{2}\sigma^2$
1.2	$XY \longrightarrow X(\sigma - \tau) + Y(\sigma - \omega) - \frac{1}{2}\sigma^2 + \frac{1}{2}\tau^2 + \frac{1}{2}\omega^2$
1.3	$XY \longrightarrow \frac{1}{2}X(\sigma - \tau) + \frac{1}{2}Y(\sigma + \tau) - \frac{1}{2}\sigma^2 + \frac{1}{2}\tau^2$
2.1	$\int^X f(u)\,du \longrightarrow X\sigma - \int^\sigma f^{-1}(u)\,du$ (f an invertable function)
2.2	$e^X \longrightarrow (X + 1)\sigma - \sigma \log \sigma$
2.3	$\lvert X\rvert\log\lvert X\rvert \longrightarrow \lvert X\rvert(\sigma + 1) - e^\sigma$
2.4	$\log\lvert X\rvert \longrightarrow X\sigma - \log\lvert\sigma\rvert$
2.5	$\frac{1}{1+\rho}\lvert X\rvert^{1+\rho} \longrightarrow X\sigma - \frac{1}{1+1/\rho}\lvert\sigma\rvert^{1+1/\rho}$ ($\rho \neq -1, 0$).
3.1	$\int^X du\left[\int^Y dv\, g(u,v)\right]^{-1}(u) \longrightarrow X\sigma - Y\tau + \int^\tau dv\left[\int^\sigma du\, g(u,v)\right]^{-1}(v)$ (If function inverses exist.)
3.2	$YF(X) \longrightarrow -X\sigma + Y\tau + \sigma F^{-1}(\tau)$ (If $((F')^{-1})'$ exists.)
3.3	$Y\int^X f(u)\,du \longrightarrow XY\sigma - Y\int^\sigma du\, f^{-1}(u)$ (If $(f^{-1})'$ exists.)
3.4	$X/Y \longrightarrow X\sigma - Y\tau + \tau/\sigma$
4.1	$\min_\alpha X_\alpha \longrightarrow \sum_\alpha X_\alpha\sigma_\alpha + C\left(\sum_\alpha \sigma_\alpha - 1\right)\lambda - \frac{C}{2}\lambda^2 + \sum_\alpha \phi(\sigma_\alpha)$ (Large C, and high-gain hysteresis-free barrier function ϕ which confines σ_α to $(0, 1)$.)
4.2	$\max_\alpha X_\alpha \longrightarrow \sum_\alpha X_\alpha\sigma_\alpha - C\left(\sum_\alpha \sigma_\alpha - 1\right)\lambda + \frac{C}{2}\lambda^2 - \sum_\alpha \phi(\sigma_\alpha)$ (Same conditions.)
4.3	$\prod_\alpha \lvert X_\alpha\rvert \longrightarrow \sum_\alpha (\sigma\tau_\alpha - \lvert X_\alpha\rvert\omega_\alpha + \omega_\alpha e^{\tau_\alpha}) + \sigma(1 - \log\sigma)$.
5.1	$\operatorname{tr} XY^T \longrightarrow \operatorname{tr} X(\sigma - \tau)^T + \operatorname{tr} Y(\sigma - \omega)^T - \frac{1}{2}\operatorname{tr}\sigma\sigma^T + \frac{1}{2}\operatorname{tr}\tau\tau^T + \frac{1}{2}\operatorname{tr}\omega\omega^T$ (All matrices, possibly sparse.)
6.1	$E[\mathbf{v}] \longrightarrow L[\dot{\mathbf{v}}\lvert\mathbf{q}] = \int dt\left(K[\dot{\mathbf{v}}\lvert\mathbf{v}, \mathbf{q}] + \frac{dE}{dt}\right)$, $\delta L/\delta\dot{\mathbf{v}}(t) = 0$ $(\partial K/\partial\dot{\mathbf{v}} = 0 \Leftrightarrow \dot{\mathbf{v}} = 0)$
6.2	$E[\mathbf{v}] \longrightarrow \hat{E}[\mathbf{q}] = \sum_i (\partial E/\partial v_i)s_i\dot{v}_i + \hat{E}_{\text{constr}}[\mathbf{q}]$ ($s_i = \pm 1$)

The variables σ, τ, ω, and λ are assumed not to occur elsewhere in the original objective. Note that each transformation may have restrictions on its applicability, in addition to the particular form it matches.

We will report experiments only with transformations 1.1, 1.2, 2.2, and 5.1 on this list. Experiments with transformations 6.1 and 6.2 will be reported in a later paper (Mjolsness & Miranker, 1990). The rest are still theoretical.

These transformations may be iterated, at the expense of creating interactions between the added variables. They can be used to reduce the nonlinearity of the interactions in a neural network, transferring such nonlinearity to single-neuron potentials or distributing it among several simpler interactions.

3. DESIGN EXAMPLES

3.1. Convolutions and Coordinate Transformations

Discrete convolutions

$$O_i = \sum_j K_{i-j} I_j$$

(where index subtraction is defined appropriately) and linear coordinate transformations

$$x'_i = \sum_j A_{ij} x_j + b_i$$

can both be expressed as sums of squared penalty terms in an objective:

$$E_{\text{conv}} = \frac{c}{2} \sum_i \left(O_i - \sum_j K_{i-j} I_j \right)^2 \qquad (37)$$

or

$$E_{\text{coord}} = \frac{c}{2} \sum_i \left(x'_i - \sum_j A_{ij} x_j + b_i \right)^2, \qquad (38)$$

which $c > 0$. (Equation (38) subsumes eqn (37).) Alternatively the convolution or coordinate change could be turned into a hard constraint by using Lagrange multiplier neurons, but those procedures still work best when a penalty term exactly like eqn (37) or eqn (38) is added to the objective (Platt & Barr, 1988; Luenberger, 1984). As they stand, these objectives expand into very expensive networks due to the spurious squaring of the matrix. That is because convolution kernels, which are usually constant but sparse, have their fanout squared; and coordinate transformations, which are usually dense but variable, have an excessive number of new (high-order) interactions created when A is squared.

Of course, eqn (38) is of the type which we know how to transform using reversed linear neurons. We obtain the modified objective

$$\dot{E}_{\text{coord}} = c \left[\sum_i \left(x'_i - \sum_j A_{ij} x_j - b_i \right) \sigma_i - \frac{1}{2} \sum_i \sigma_i^2 \right] \qquad (39)$$

which doesn't square A. If A is constant then there is no order reduction, since both E_{coord} and $\dot{E}_{\text{coord}}$ are second order, but there are fewer connections unless A is also dense.

An alternative objective, not using reversed neurons, is also available for convolutions and coordinate system transformations. The objective

$$\dot{E}_{\text{coord}} = \frac{c}{2} \sum_{ij} A_{ij} (x'_i - b_i - x_j)^2 \qquad (40)$$

is minimal with respect to x' when

$$x'_i = \sum_j A_{ij} x_j \Big/ \sum_j A_{ij} + b_i. \qquad (41)$$

This type of dynamic normalization may be desirable, or if A is constant and already normalized then it does not hurt. Equation (40) also preserves any sparseness of A, and does not square the matrix.

3.2. Random Dot Matching

Here the problem begins with two inputs: a planar pattern of n dos specified by their independent random positions $\mathbf{x}_i$ within a unit square, and a pattern of $m << n$ dots specified by their positions $\mathbf{y}_a$. The $\mathbf{y}_a$ are generated by randomly selecting m of the $\mathbf{x}$ dots, independently perturbing their positions by random displacements of 1/10 the size of the square, and globally shifting all m dots by a translation vector Δ. The problem is to reconstruct Δ from $\{\mathbf{x}_i\}$ and $\{\mathbf{y}_a\}$, by consistently matching the dots. Since the match is parameterized by a few geometric parameters, this is really an image "registration" problem.

A simple objective for this task is

$$E_{\text{dots}}[\Delta] = -\sum_i \sum_a \exp(-|\mathbf{x}_i - \mathbf{y}_a - \Delta|^2 / 2K^2), \qquad (42)$$

where the search width K is to be set initially to the width of the square (1.0) and gradually decreased down to the size of the geometric noise (0.1) as the optimization proceeds, in order to find better local minima. This objective, and the gradual change in K, is quite similar to that of the elastic net approach to the traveling salesman problem (Durbin & Willshaw, 1987). Now E_{dots} may be transformed to remove the exponential from the interactions:

$$\dot{E}_{\text{dots}}[\Delta, \tau] = \frac{1}{2K^2} \sum_{ia} |\mathbf{x}_i - \mathbf{y}_a - \Delta|^2 \tau_{ia} + \sum_{ia} \tau_{ia} (\log \tau_{ia} - 1) \qquad (43)$$

with dynamics

$$\begin{aligned} \dot{\Delta} &= (1/K^2) \sum_{ia} \tau_{ia} (\mathbf{x}_i - \mathbf{y}_a - \Delta), \\ \tau_{ia} &= \exp \omega_{ia}, \\ \dot{\omega}_{ia} &= -\omega_{ia} - (1/2K^2) |\mathbf{x}_i - \mathbf{y}_a - \Delta|^2. \end{aligned} \qquad (44)$$

We experiment with $n = 30$ and $m = 6$. In Figure 3 we have shown a contour map of E_{dots}, which is to be minimized, along with the projection of a $K = .2$ trajectory onto the Δ plane. The initial condition came from partially relaxing the net at $K = .5$ first, where the objective is unimodal. The $K = .2$ problem is somewhat more difficult than incrementally updating a solution in response to a small change in K, but the network found the right answer. For $n = 30$ and $m = 6$, some random patterns of dots would require several large-K minima to be tracked to small K for correct operation, but this defect of the original objective (42) did not arise for the case shown here (or 13 out of 15 other cases we examined) and is not relevant to the validity of the transformation.

The main numerical drawback of the 180 exponential-taking neurons in this net is that small time steps may be required. In using the Runge-Kutta method for solving eqn (44) the stepsize had to be $\Delta t = .0003$ near the starting point, even though eventually it could be increased to .003.

3.3 Graph Matching and Quadratic Match

Consider the following objective for inexact graph-matching (Hopfield & Tank, 1986; c.f. von der Malsburg & Bienenstock, 1986):

$$E_{graph} = -c_1 \sum_{\alpha\beta ij} G_{\alpha\beta} g_{ij} M_{\alpha i} M_{\beta j} + c_2 \sum_{\alpha} \left(\sum_{i} M_{\alpha i} - 1 \right)^2 + c_2 \sum_{i} \left(\sum_{\alpha} M_{\alpha i} - 1 \right)^2 + c_3 \sum_{\alpha i} M_{\alpha i}(1 - M_{\alpha i}) + \sum_{\alpha i} \int^{M_{\alpha i}} dx\, g^{-1}(x), \tag{45}$$

where G and g are connection matrices (each entry is zero or one) for two graphs, and $M_{\alpha i}$ is one when node α of G maps to node i of g, and zero otherwise.

The problem may be generalized slightly to quadratic matching by replacing the $GgMM$ term with

$$\sum_{\alpha\beta ij} G_{\alpha\beta} g_{ij} A_{\alpha i} B_{\beta j} \tag{46}$$

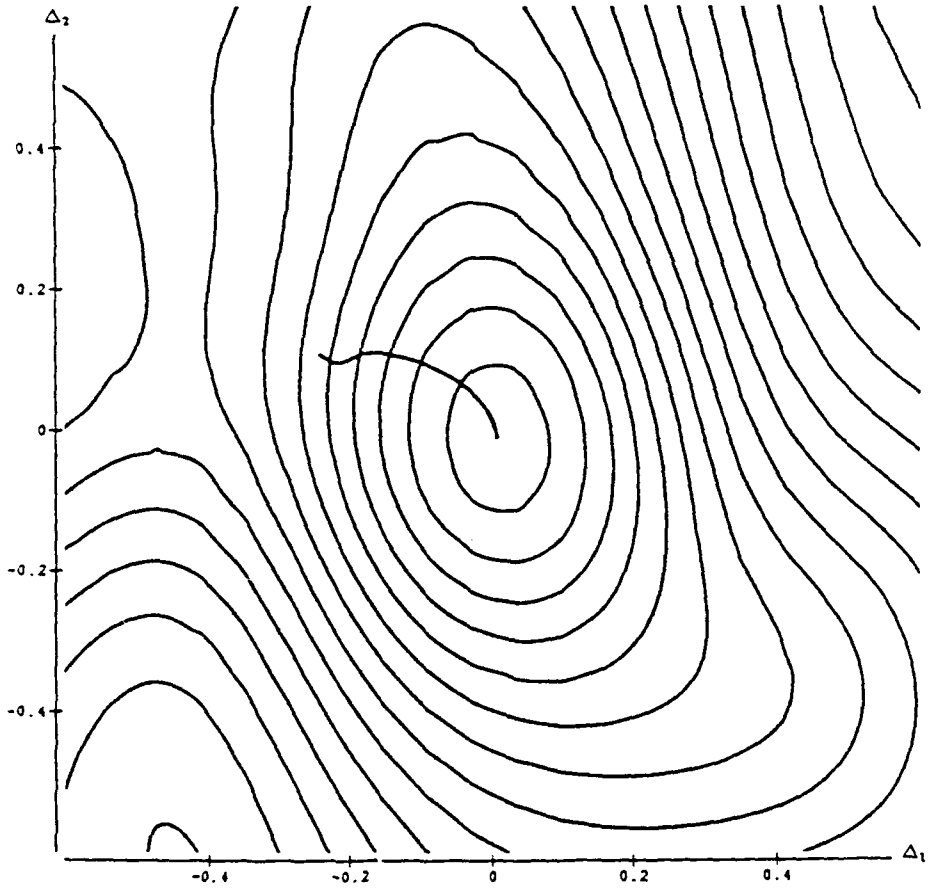


FIGURE 3. Random dot matching network. *Trajectory of net evolution equations* (44) projected to the $\bar{\Delta}$ plane, superposed on a contour plot of E_{dots}. K was .2. The starting point was obtained by a partial relaxation of the same net for $K = .5$, for which the objective has a single local minimum, starting from $\bar{\Delta} = (1, 1)$ and $\bar{\omega} = \bar{0}$. (Further relaxation at $K = .5$ would result in an initial $\bar{\Delta}$ even closer to the $K = .2$ minimum). The correct answer is $\bar{\Delta} = \bar{0}$.

and altering the other constraints to reflect the fact that $A = B$. What we have to say about graph matching will apply equally well to this generalization.

The $GgMM$ term is superficially the expensive one since it involves four sums. If each graph is constant, there are $\mathcal{O}(N)$ nodes in each graph, and both graphs have fanout f, then the number of monomial synapses is $\mathcal{O}(N^2f^2)$. We can reduce this to $\mathcal{O}(N^2f)$. Also if one of G or g is variable, with N nodes in the variable graph and m in the constant graph, as in the "Frameville" networks of (Mjolsness et al., 1989a), and both graphs are represented densely, then the number of synapses is reduced from $\mathcal{O}(N^2mf)$ to $\mathcal{O}(N^2m + Nmf)$. The reduction uses linear interneurons, both reversed and normal:

$$\begin{aligned} E_1 &= -c_1 \sum_{\alpha\beta ij} G_{\alpha\beta} g_{ij} M_{\alpha i} M_{\beta j} \\ &= -\frac{c_1}{2} \sum_{\beta i} \left[\left(\sum_{\alpha} G_{\alpha\beta} M_{\alpha i} + \sum_{j} g_{ij} M_{\beta j} \right)^2 \right. \\ &\quad \left. - \left(\sum_{\alpha} G_{\alpha\beta} M_{\alpha i} \right)^2 - \left(\sum_{j} g_{ij} M_{\beta j} \right)^2 \right], \\ \dot{E}_1 &= -c_1 \left[\sum_{\alpha ij} G_{\alpha\beta} M_{\alpha i} \sigma_{\beta j} + \sum_{\beta ij} g_{ij} M_{\beta j} \sigma_{\beta i} \right. \\ &\quad - \sum_{\alpha ij} G_{\alpha\beta} M_{\alpha i} \tau_{\beta j} - \sum_{\beta ij} g_{ij} M_{\beta j} W_{\beta i} \\ &\quad \left. + \frac{1}{2} \sum_{\beta i} (-\sigma_{\beta i}^2 + \tau_{\beta i}^2 + w_{\beta i}^2 \right. \end{aligned} \tag{47}$$

E_1 and $\dot{E}_1$ are illustrated in Figure 4.

The reduced graph-matching network works in simulation, for five out of six small ($N = 10$ nodes) hand-designed graphs with low fanout (from 1.8 to 2.5). The sixth case is not solved by the original, untransformed network either. The parameters we used were

$N = 10$	$c_1 = 1.0$	$c_2 = 1.0$	$c_3 = 0.5$
$g_0(M) = 20$	$g_0(\sigma) = 1.0$	$g_0(\text{WTA}) = 10.0$	$r_\sigma = 1$
$\Delta t = .004$	sweeps = 1 000		

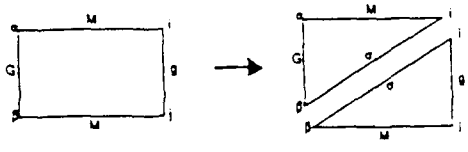


FIGURE 4. Graph matching networks. E_1 is a sum over the indices α, β, i, and j, which are connected by neurons (line segments) in the shape of a "rectangle". This objective can be transformed into $\dot{E}_1$ which is a sum of triangles, while preserving fixpoints. The triangles are obtained from the rectangle by introducing linear interneurons along a diagonal, as shown. Only three indices are summed over, resulting in a less costly network.

and for the original network:

$N = 10$	$c_1 = 1.0$	$c_2 = 1.0$	$c_3 = 0.5$
$g_0(M) = 20$	$g_0(\sigma) = 1.0$	$g_0(\text{WTA}) = 10.0$	$\Delta t = .004$
sweeps = 1 000			

Here $g_0(M)$ is the gain $g'(0)$ of the transfer function $g(M)$ for M, and similarly $g_0(\sigma)$ is the gain for the linear neurons that were introduced through transforming E_1, namely σ, τ, and ω. Also $g_0(\text{WTA})$ is the gain for the infinitely fast linear neurons which were used in both reduced and control experiments to implement the WTA constraints; this parameter effectively multiplies c_1. sweeps is the number of iterations of the forward Euler method used in simulating the continuous update equations. Each iteration advanced the time coordinate by Δt.

There is only a little parameter-tuning involved here, concentrated on r_σ, Δt, and sweeps. The product $\Delta t \times \max(r_\sigma, r_M = 1)$ should be held fixed to maintain constant resolution in the discrete simulation of continuous update equations. But holding Δt and the other parameters fixed at the quoted values, the rate parameter r_σ can be varied from unity to 100 without altering the network convergence time, measured in sweeps, by more than 30% or so; network performance remains the same in that the same 5 out of 6 graphs are correctly matched. This would suggest, and other experiments confirm, that for low r there is some room for increasing Δt and decreasing sweeps ($r = 10$, $\Delta t = .016$, and sweeps = 300, respectively); this saves time whether time is measured in simulation sweeps or in circuit time constants.

3.4 Sorting

Sorting may be described in a manner very similar to graph matching. One requires a permutation matrix M which sorts the inputs into increasing order, so all terms in the objective remain the same except for $GgMM$. The objective becomes

$$\begin{aligned} E_{\text{sort}} &= -c_1 \sum_{ij} M_{ij} x_i y_j + c_2 \sum_{i} \left(\sum_{j} M_{ij} - 1 \right)^2 \\ &\quad + c_2 \sum_{j} \left(\sum_{i} M_{ij} - 1 \right)^2 + c_3 \sum_{ij} M_{ij}(1 - M_{ij}) \\ &\quad + \sum_{ij} \int^{M_{ij}} dx\, g^{-1}(x). \end{aligned} \tag{48}$$

Here x_i are a set of input numbers to be sorted, and y_j are a constant set of numbers in increasing order (e.g., $y_j = j$). M will become that permutation matrix which maximizes the inner product of x and y (i.e., maps the largest x_i to the largest y_j, the next largest x_i to the next largest y_j, and so on).

After using the winner-take-all reduction on the row and column constraints of M, this network has $\mathcal{O}(N^2)$ connections and neurons. One cannot do better without reducing the number of match neurons M. But M is sparse at any acceptable answer, so it may be possible to keep it sparse throughout the time the network is running by using a different encoding of the matrix. For example, one might encode indices i, j, or both in a binary representation as would be done in an ordinary computer program for sorting, in which one commonly uses the binary representation of pointers to represent a sparse graph $M_{ij} \in \{0, 1\}$; alternatively if M is always a permutation one can represent it by a one-dimensional array of binary addresses $j[i]$. The resulting objectives generally still have $\mathcal{O}(N^2)$ connections (monomial interactions), but a well-chosen matrix encoding, supplemented by suitable reversed neurons, can drastically reduce the number of connections.

In Appendix B it is shown that any permutation matrix M of size $L^2 \times L^2$ can be represented in the following form:

$$M_{i_1 i_2 \cdot j_1 j_2} = \sum_{k_1 k_2} A^{(1)}_{i_1 i_2 \cdot k_1} A^{(2)}_{i_2 \cdot k_1 k_2} \tilde{A}^{(1)}_{j_1 j_2 \cdot k_1} \tilde{A}^{(2)}_{j_2 \cdot k_1 k_2}, \qquad (49)$$

where $i \in \{1, \ldots N = L^2\}$, $i_k \in \{1, \ldots \sqrt{N} = L\}$, and $i = i_1 L + i_2 \equiv (i_1, i_2)$, and where two constraints apply to each nonsquare matrix:

$$\begin{array}{cccc} \sum_{j_1} A^{(1)}_{i_1 i_2 \cdot j_1} = 1 & \left| \sum_{i_2} A^{(2)}_{i_2 \cdot j_1 j_2} = 1 \right. & \left\| \sum_{j_1} \tilde{A}^{(1)}_{j_1 j_2 \cdot j_1} = 1 \right. & \left| \sum_{i_2} \tilde{A}^{(2)}_{i_2 \cdot j_1 j_2} = 1 \right. \\ \sum_{j_1} A^{(1)}_{i_1 i_2 \cdot j_1} = 1 & \left| \sum_{i_2} A^{(2)}_{i_2 \cdot j_1 j_2} = 1 \right. & \left\| \sum_{j_1} \tilde{A}^{(1)}_{j_1 j_2 \cdot j_1} = 1 \right. & \left| \sum_{i_2} \tilde{A}^{(2)}_{i_2 \cdot j_1 j_2} = 1 \right. \end{array} \qquad (50)$$

The matrix form (49) contains only $4N^{3/2}$ variables, and is our proposed encoding of M.

As explained in Appendix C, eqn (49) is a coarse version of another expression for M which contains $\mathcal{O}(N \log N)$ variables. That expression codes index pairs (i, j) using the "Butterfly" connection topology that arises in the fast Fourier transform (FFT) and in many other parallel algorithms. The advantage of the Butterfly is that it allows one to make a gradual transition from one space (e.g., index i) to another (index j). There has been little success in transforming objectives based on the much less gradual binary or base-b encoding of i and M_{ij}.[2] An example similar to eqn (49) is the base $\sqrt{N}$ code obtained by listing all N links in the permutation matrix, indexed by k, and encoding their starting and ending locations as $i = (i_1, i_2)$ and $j = (j_1, j_2)$. Then

$$M_{i_1 i_2 \cdot j_1 j_2} = \sum_k A^{(1)}_{i_1 \cdot k} A^{(2)}_{i_2 \cdot k} \tilde{A}^{(1)}_{j_1 \cdot k} \tilde{A}^{(2)}_{j_2 \cdot k} \qquad (51)$$

subject to obvious constraints on the A's.

Since any permutation matrix can be expressed using eqn (49), any (approximately quadratic) local minimum of $E_{sort}(M)$ (eqn (48)), for which M is sufficiently close to being a permutation matrix, should be a local minimum of $E_{sort}(A^{(1)}, A^{(2)}, \tilde{A}^{(1)}, \tilde{A}^{(2)})$; but there may be local minima with respect to A and $\tilde{A}$ which would be unstable in the larger M space. Thus, making the substitution (49) into eqn (48) may expand the set of fixed points, or alter it entirely if the original objective is not yet tuned to produce permutation matrices. By contrast, the objective function transformations used heretofore have exactly preserved the set of fixed points. We will try it out anyway, in order to get a low-cost net, and we will observe whether and how well it sorts.

The problem now is to reduce the number of monomial interactions of $\mathcal{O}(N^{3/2})$. This is easy for quadratic penalty terms corresponding to the constraints (50), which consist of $8N$ winner-take-all constraints each involving $N^{1/2}$ variables. The remaining interaction $-c \sum Mxy$ can be reduced in two stages: substitute $M = A\tilde{A}^T$ with no reduction in connection costs; then substitute the $\mathcal{O}(N^{3/2})$ forms for A and $\tilde{A}$.

Thus we may replace $E_1 = -c_1 \sum_{ij} M_{ij} x_i y_j$ with

$$\begin{aligned} E_1(A, \tilde{A}) &= -c_1 \sum_{ijk} A_{ik} \tilde{A}_{jk} x_i y_j \\ &= -\frac{c_1}{2} \left[\sum_k \left(\sum_i A_{ik} x_i + \sum_j \tilde{A}_{jk} y_j \right)^2 \right. \\ &\quad \left. - \sum_k \left(\sum_i A_{ik} x_i \right)^2 - \sum_k \left(\sum_j \tilde{A}_{jk} y_j \right)^2 \right] \\ \dot{E}_1(A, \tilde{A}, a, b, \tilde{b}) &= -c_1 \left[\sum_k (a_k - b_k) \sum_i A_{ik} x_i \right. \\ &\quad + \sum_k (a_k - \tilde{b}_k) \sum_j \tilde{A}_{jk} y_j \\ &\quad \left. + \tfrac{1}{2} \sum_k \left(-a_k^2 + b_k^2 + \tilde{b}_k^2 \right) \right] \end{aligned} \qquad (52)$$

which may be interpreted as four interacting sorting problems, with linear interneurons a interpolating between x and y and with reversed neurons b and $\tilde{b}$ cancelling echos as in eqn (15). So far there is no reduction in number of neurons or connections.

[2] A partial exception is the load-balancing network of Fox and Furmansky (1988), in which the crucial "histogram" may be understood as a set of reversed linear interneurons which simplify their load-balancing objective. But the result is a virtual neural net, not a statically connected circuit.

If we substitute the special forms for A and $\tilde{A}$, we find

$$\begin{aligned}
\dot{E}_1 &= -c_1 \Bigg[\sum_{i_1 i_2} \sum_{k_1 k_2} x_i A^{(1)}_{i_1 k_1} A^{(2)}_{i_2 k} (a_k - b_k) \\
&\quad + \sum_{j_1 j_2} \sum_{k_1 k_2} y_j \tilde{A}^{(1)}_{j_1 k_1} \tilde{A}^{(1)}_{j_2 k} (a_k - \tilde{b}_k) \\
&\quad - \tfrac{1}{2} \sum_k a_k^2 + \tfrac{1}{2} \sum_k b_k^2 + \tfrac{1}{2} \sum_k \tilde{b}_k^2 \Bigg], \\
&= -c_1 \Bigg[\tfrac{1}{2} \sum_{i_2 k_1} \Bigg[\Bigg(\sum_{i_1} A^{(1)}_{i_1 i_2 \cdot k_1} x_i + \sum_{k_2} A^{(2)}_{i_2 \cdot k_1 k_2} (a_k - b_k) \Bigg)^2 \\
&\quad - \Bigg(\sum_{i_1} A^{(1)}_{i_1 i_2 \cdot k_1} x_i \Bigg)^2 - \Bigg(\sum_{k_2} A^{(2)}_{i_2 \cdot k_1 k_2} (a_k - b_k) \Bigg)^2 \Bigg] \\
&\quad + \tfrac{1}{2} \sum_{j_2 k_1} \Bigg[\Bigg(\sum_{j_1} \tilde{A}^{(1)}_{j_1 j_2 \cdot k_1} y_j + \sum_{k_2} \tilde{A}^{(2)}_{j_2 \cdot k_1 k_2} (a_k - \tilde{b}_k) \Bigg)^2 \\
&\quad - \Bigg(\sum_{j_1} \tilde{A}^{(1)}_{j_1 j_2 \cdot k_1} y_j \Bigg)^2 - \Bigg(\sum_{k_2} \tilde{A}^{(2)}_{j_2 \cdot k_1 k_2} (a_k - \tilde{b}_k) \Bigg)^2 \Bigg] \\
&\quad - \tfrac{1}{2} \sum_k a_k^2 + \tfrac{1}{2} \sum_k b_k^2 + \tfrac{1}{2} \sum_k \tilde{b}_k^2 \Bigg],
\end{aligned} \tag{53}$$

and finally

$$\begin{aligned}
\dot{E}_1 &= -c_1 \Bigg[\sum_{i k_1} A^{(1)}_{i_1 i_2 \cdot k_1} x_i (\sigma_{i_2 k_1} - \tau_{i_2 k_1}) \\
&\quad + \sum_{i_2 k} A^{(2)}_{i_2 \cdot k_1 k_2} (a_k - b_k)(\sigma_{i_2 k_1} - \omega_{i_2 k_1}) \\
&\quad - \tfrac{1}{2} \sum_{i_2 k_1} \sigma^2_{i_2 k_1} + \tfrac{1}{2} \sum_{i_2 k_1} \tau^2_{i_2 k_1} + \tfrac{1}{2} \sum_{i_2 k_1} \omega^2_{i_2 k_1} \\
&\quad + \sum_{j k_1} \tilde{A}^{(1)}_{j_1 j_2 \cdot k_1} y_j (\tilde{\sigma}_{j_2 k_1} - \tilde{\tau}_{j_2 k_1}) + \sum_{j_2 k} \tilde{A}^{(2)}_{j_2 \cdot k_1 k_2} \\
&\quad \times (a_k - \tilde{b}_k)(\tilde{\sigma}_{j_2 k_1} - \tilde{\omega}_{j_2 k_1}) \\
&\quad - \tfrac{1}{2} \sum_{j_2 k_1} \tilde{\sigma}^2_{j_2 k_1} + \tfrac{1}{2} \sum_{j_2 k_1} \tilde{\tau}^2_{j_2 k_1} + \tfrac{1}{2} \sum_{j_2 k_1} \tilde{\omega}^2_{j_2 k_1} \\
&\quad - \tfrac{1}{2} \sum_k a_k^2 + \tfrac{1}{2} \sum_k b_k^2 + \tfrac{1}{2} \sum_k \tilde{b}_k^2 \Bigg]
\end{aligned} \tag{54}$$

which has $\mathcal{O}(N^{3/2})$ neurons and connections.

In Appendix C we show how to extend this result to a series of successively cheaper approximations of the original sorting network, down to $\mathcal{O}(N \log N)$ neurons and connections.

3.5. Sorting: Experiments

The $\mathcal{O}(N^{3/2})$ sorting network only sorts in an approximate way. The reversed neurons work correctly at finite r, which is nontrivial since they are connected to each other, but the encoding scheme is prone to trapping by local minima. We used the following parameter values for the $\mathcal{O}(N^{3/2})$ sorting network:

$N = 16$	$c_1 = 0.6$	$c_2 = 6.0$	$c_3 = 0$
$g_0(A) = 20$	$r_A = 1$	$r_\sigma = 3$	$\Delta t = .01$
sweeps = 20 000			
$N = 25$	$c_1 = .44$	$c_2 = 6.0$	$c_3 = 0$
$g_0(A) = 20$	$r_A = 1$	$r_\sigma = 3$	$\Delta t = .01$
sweeps = 20 000			

and for the $\mathcal{O}(N^2)$ network:

$N = 16$	$c_1 = 0.6$	$c_2 = 6.0$	$c_3 = 0$
$g_0(A) = 20$	$\Delta t = .01$	sweeps = 5 000	
$N = 25$	$c_1 = .44$	$c_2 = 6.0$	$c_3 = 0$
$g_0(A) = 20$	$\Delta t = .01$	sweeps = 5 000	

As in the graph-matching example, most of the parameter-tuning was concentrated on r_σ, Δt, and sweeps. Here, r_σ is the rate parameter appearing in the update equation (7), and it applies to neurons a, b, $\tilde{b}$, σ, $\tilde{\sigma}$, τ, $\tilde{\tau}$, ω, $\tilde{\omega}$. Likewise r_A applies to the update equations for A and $\tilde{A}$. As in eqn (48), c_1 multiplies the strength of the permuted inner product of x and y in the objective. Also c_2 is the strength of the syntax

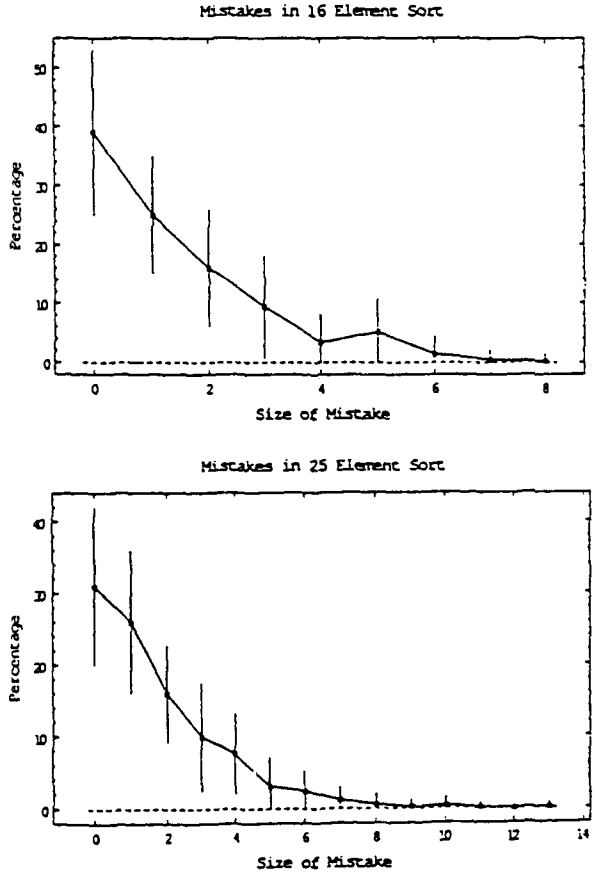


FIGURE 5. Histogram of placement errors. Sorting network with butterfly encoding, $\mathcal{O}(N^{3/2})$ connections. (a) Size $N = 16$. Average and standard deviation (upper or lower half of an error bar) for 62 runs. (b) Size $N = 25$. Average and standard deviation for 39 runs.

constraints, and c_3 is the strength of a term penalizing intermediate neuron values. $g_0(A)$ is the gain $g'(0)$ of the transfer function $g(A)$ for A and $\tilde{A}$, which obeyed steepest-descent dynamics. The constant y values are $y_1 = -(N-1)/2$, $y_2 = -(N-3)/2$, $\ldots$ $y_{N-1} = (N-3)/2$, $y_N = (N-1)/2$. sweeps is the number of iterations of the forward Euler method used in simulating the continuous update equations.

For input size $N = 16$ we find an average placement error of 1.4 out of 16 possible output places. Eight would be random. For $N = 25$, which is the first size (with integral $\sqrt{N}$) for which there are fewer neurons in the asymptotically smaller network, the average placement error is 1.7 out of 25 places. The errors can be characterized by a histogram showing the frequency with which placement errors of different sizes occur. (The size of a placement error is the difference, in the permuted output vector, between the desired and actual positions of an element.) Histograms for $N = 16$ and $N = 25$ are presented in Figure 5 and they show that small mistakes are far more likely than large ones, and that the frequency falls off as roughly the -2.1 (respectively, -2.0) power of the size of the placement error, with correlation $r = .90$ (.93). In addition, 17.7% (respectively, 21%) of the experimental runs failed to meet our convergence criterion, which was that exactly one element in each row and column of the computed matrix M must have a value greater than .5.

Iterating the sort can improve the score marginally, but not to the perfect sorts achievable with the $\mathcal{O}(N^3)$ network at these sizes.

4. DISCUSSION AND CONCLUSION

Although much research now focuses on expressing new computational problems using objective functions and then deriving neural networks which solve them, we would like to suggest that when such efforts are successful there may be an additional advantage to be obtained. If the solution can be regarded as a novel algebraic transformation of an unremarkable objective, then the transformation may also be immediately applicable to other objectives and problems. This provides a kind of *reuse* of neural net design effort which is fundamentally more flexible than the reuse of modules or components as practiced in electronic design (Heinbuch, 1988), and in neural net learning (e.g., Mjolsness et al., 1989b). The transformational approach has also proved useful in VLSI design [e.g., transforming a program into a microprocessor (Martin, Burns, Lee, Borkovic & Hazewindus, 1989)].

We have shown that there are algebraic transformations of objective functions which not only preserve the set of fixpoints of the resulting analog neural network, but alter the number and nature of interactions requiring physical connections between different neurons. Some of these transformations require the network to find a saddle point rather than a minimum of the objective, but that can be arranged. Others provide control over the dynamics by which an objective is extremized. A set of such transformations was derived, together with their conditions of validity.

Several design examples were given, along with experimental results to show convergence with reasonable parameter values. Reduced-cost designs were presented for convolution and linear coordinate transformation. A reduced network for sorting converged to approximately correct answers despite the use of an illegitimate transformation which introduced spurious fixpoints. This design also involved legitimate but repeated, interacting product-reduction transformations. Transformed networks for quadratic matching and for registration of random dot patterns were simulated without difficulty.

REFERENCES

Arrow, K. J., Hurwicz, L., & Uzawa, H. (Eds.). (1958). *Studies in linear and nonlinear programming*. Palo Alto, CA: Stanford University Press.

Benes, V. E. (1965). *Mathematical theory of connecting networks and telephone traffic* (p. 113). New York: Academic Press.

Courant, R., & Hilbert, D. (1962). *Methods of mathematical physics* (Vol. II, pp. 32–39). New York: John Wiley and Sons.

Durbin, R., & Willshaw, D. J. (1987). An analogue approach to the travelling salesman problem using an elastic net method. *Nature*, 326, 689–691.

Feldman, J. A. (1982). Dynamic connections in neural networks. *Biological Cybernetics*, 46, 27–39.

Fox, G. C., & Furmansky, W. (1988). *Load balancing loosely synchronous problems with a neural network* (Technical Report *C³ P363B*). California Institute of Technology.

Gindi, G., Gmitro, A., & Parthasarathy, K. (1987). Winner-take-all networks and associative memory: Analysis and optical realization. In *Proc. of First International Conference on Neural Networks* (Vol. III, pp. 607–614). New York: IEEE.

Grossberg, S. (1988). Nonlinear neural networks: Principles, mechanisms, and architectures. *Neural Networks*, 1, 17–61.

Heinbuch, D. V. (Ed.). (1988). *CMOS3 cell library*. Reading, MA: Addison-Wesley.

Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences USA*, 81, 3088–3092.

Hopfield, J. J., & Tank, D. W. (1985). "Neural" computation of decisions in optimization problems. *Biological Cybernetics*, 52, 141–152.

Hopfield, J. J., & Tank, D. W. (1986). Collective computation with continuous variables. In *Disordered systems and biological organization* (pp. 155–170). Berlin: Springer-Verlag.

Koch, C., Marroquin, J., & Yuille, A. (1986). Analog "neuronal" networks in early vision. *Proceedings of the National Academy of Sciences USA*, 83, 4263–4267.

Luenberger, D. G. (1984). *Linear and nonlinear programming*. Reading, MA: Addison-Wesley.

Martin, A. J., Burns, S. M., Lee, T. K., Borkovic, D., & Ha-

zewindus, P. J. (1989). The design of an asynchronous microprocessor. In *Proc. Decennial Caltech Conference on VLSI* (pp. 351–373). Cambridge, MA: MIT Press.

Mjolsness, E. (1987). Control of attention in neural networks. In *Proc. of First International Conference on Neural Networks* (Vol. II, pp. 567–574). New York: IEEE.

Mjolsness, E., Gindi, G., & Anandan, P. (1989a). Optimization in model matching and perceptual organization. *Neural Computation*, 1, 218–229.

Mjolsness, E., & Miranker, W. (1990, June.). Research Report YALEV/DCS/RR-797, Yale Computer Science Department.

Mjolsness, E., Sharp, D. H., & Alpert, B. K. (1989b). Scaling, machine learning, and genetic neural nets. *Advances in Applied Mathematics*, 10, 137–163.

Moody, J. (1989). *Optimal architectures and objective functions for associative memory* (Technical Report TR668). Yale University Department of Computer Science.

Platt, J. C., & Barr, A. H. (1987). Constrained differential optimization. In D. Z. Anderson (Ed.), *Neural information processing systems*. New York: American Institute of Physics.

Platt, J. C., & Barr, A. H. (1988). Constraint methods for flexible models. *Computer Graphics*, 22(4). Proceedings of SIGGRAPH '88, pp. 279–288.

Rumelhart, D. E., Hinton, G. E., & McClelland, J. L. (1986a). A general framework for parallel distributed processing. In *Parallel distributed processing* (pp. 73–74). Cambridge, MA: MIT Press.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986b). Learning internal representations by error propagation. In *Parallel distributed processing* (pp. 318–362). Cambridge, MA: MIT Press.

Simic, P. (1989). Personal communication.

Sivilotti, M. A., Mahowald, M. A., & Mead, C. A. (1987). Real-time visual computations using analog CMOS processing arrays. In *Advanced Research in VLSI: Proceedings of the 1987 Stanford Conference*. Cambridge, MA: MIT Press.

Tank, D. W., & Hopfield, J. J. (1986). Simple "neural" optimization networks: An a/d converter, signal decision circuit, and a linear programming circuit. *IEEE Transactions on Circuits and Systems*, CAS-33, 533–541.

Utans, J., Gindi, G., Mjolsness, E., & Anandan, P. (1989). *Neural networks for object recognition within compositional hierarchies: Initial experiments* (Technical Report Center for Systems Science Report No. 8903). Yale University Department of Electrical Engineering.

von der Malsburg, C., & Bienenstock, E. (1986). Statistical coding and shortterm synaptic plasticity: A scheme for knowledge representation in the brain. In *Disordered systems and biological organization* (pp. 247–252). Berlin: Springer-Verlag.

von Neumann, J., & Morgenstern, O. (1953). *Theory of games and economic behavior*. Princeton, NJ: Princeton University Press.

APPENDIX A. REDUCING $YF(X)$

The problem is to use eqn (21) to reduce expressions of the form $YF(X)$. We will derive two versions: the first, from stopping after the first step in the derivation of (21); the second, from carrying the transformation all the way through.

Now

$$\begin{aligned} Y\int^x du\, f(u) &= \int^x du\left[\int^\gamma dv\, g(u,v)\right]^{-1}(u) \\ \Rightarrow \quad Yf(X) &= \left[\int^\gamma dv\, g(u,v)\right]^{-1}(X) \\ \Rightarrow \int^\gamma dv\, g(X,v) &= f^{-1}(X/Y) \\ \Rightarrow \quad g(u,v) &= (d/dv)f^{-1}(u/v) \\ &= -(u/v^2)(f^{-1})'(u/v). \end{aligned} \tag{55}$$

To use the first step in the derivation of eqn (21), we must evaluate

$$\int^\alpha du \int^\gamma dv\, g(u,v) = \int^\alpha du\, f^{-1}(u/v) = Y\int^{\alpha/\gamma} du\, f^{-1}(u)$$

so

$$Y\int^x f(u)\,du \longrightarrow XY\sigma - Y\int^\alpha du\, f^{-1}(u) \tag{56}$$

since σ may be rescaled by Y. This shows that both sides of the transformation (16) may be multiplied by any expression Y.

To carry through transformation (21) we must calculate

$$\begin{aligned} &\int^\tau dv\left[\int^\alpha du\, g(u,v)\right]^{-1}(v) \\ &\quad = \int^\tau dv\left[\int^\alpha du(-u/v^2)(f^{-1})'(u/v)\right]^{-1}(v) \\ &\quad = \int^\tau dv\left[-\int^{\alpha/v} du\, u(f^{-1})'(u)\right]^{-1}(v) \\ &\quad = \int^\tau dv\left[-\int^{f^{-1}} (\sigma/v)\,dz\, f(z)\right]^{-1}(v) \\ &\quad = \int^\tau dv[-F(f^{-1}(\sigma/v))]^{-1}(v) \\ &\quad = \int^\tau dv[\sigma/f(F^{-1}(-v))] = -\sigma\int^{F^{-1}(-\tau)} dw \\ &\quad = -\sigma F^{-1}(-\tau) \end{aligned} \tag{57}$$

from which we deduce that

$$YF(X) \longrightarrow -X\sigma + Y\tau + \sigma F^{-1}(\tau). \tag{58}$$

(τ and σ have been rescaled by -1.) The algebra can be checked by optimizing with respect to σ. Note that the derivations of (56) and (58) assume that $f = F$ is invertable and that $f^{-1} = (F')^{-1}$ is differentiable (i.e., $((F')^{-1})'$ exists).

APPENDIX B. BUTTERFLY NETWORKS

B.1. Back-to-back Butterflies

By a recursive induction argument (Benes, 1965), any permutation matrix of size $N = 2^n$ (n an integer) can be expressed by setting switches in two back-to-back butterfly networks independently, as shown in Figure 6. If we label the corresponding connection matrices A and $\hat{A}$, then the entire network represents a permutation matrix M in the form of a matrix product $M_{ij} = (A\hat{A}^T)_{ij} = \sum_{k,l} A_{ik}\hat{A}_{jl}$, with A and $\hat{A}$ being of the special "butterfly" form. Butterfly networks are best analyzed by introducing binary notation for all indices, for example,

$$\begin{aligned} i &\longrightarrow (p_1, \ldots p_n) \sim p_1 \ldots p_n, \\ j &\longrightarrow (q_1, \ldots q_n) \sim q_1 \ldots q_n. \end{aligned} \tag{59}$$

In this notation, the outer column of switches in the A butterfly has the form

$$B^{(1)}_{p_1 \ldots p_{n-1} q_1} \in [0, 1] \tag{60}$$

with constraints

$$\sum_{p_1} B^{(1)}_{p_1 \ldots p_{n-1} q_1} = 1 \quad \text{and} \quad \sum_{q_1} B^{(1)}_{p_1 \ldots p_{n-1} q_1} = 1 \tag{61}$$

as illustrated by the butterfly ($\bowtie$) highlighted in Figure 6. Likewise the lth column of switches has the form

$$\begin{aligned} &B^{(l)}_{p_1 \ldots p_{n-1} q_1 \ldots q_{l-1}} \in [0, 1] \\ &\qquad \text{with} \sum_{p_l} B^{(l)}_{p_1 \ldots p_{n-1} q_1 \ldots q_{l-1}} = 1 = \sum_{q_l} B^{(l)}_{p_1 \ldots p_{n-1} q_1 \ldots q_{l-1}}, \end{aligned} \tag{62}$$

for $1 \le l \le n$. With constraints, each of the $\log N$ layers contains $N/2$ bits, which is one reason that $\hat{A}$ is also needed to specify an entire permutation matrix.

The entire permutation matrix is obtained by finding all the possible paths through the network from i to j, of which there is only one since each stage irrevocably decides one bit of j. This is

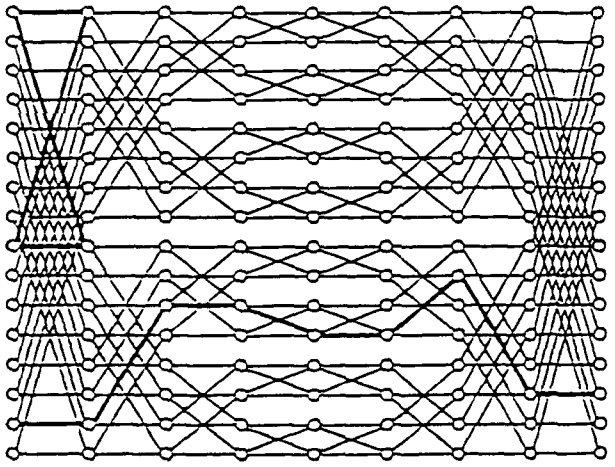


FIGURE 6. Butterfly switching networks. Any permutation of $N = 2^n$ elements can be represented by appropriately setting the switches in a pair of back-to-back butterfly switching networks, as can be shown recursively by induction on n. Highlighted: one 2 × 2 permutation matrix or "butterfly", and one path through the entire switching network.

equivalent to a conjunction of switch settings:

$$A_{\bar{i}} = \prod_{l=1}^{n} B^{(l)}_{p_1 \cdots p_n q_1 \cdots q_l}. \tag{63}$$

It is easy to check that this is a permutation matrix, using the constraints on $A^{(l)}$. (In what follows the terms of a product Π do not commute because they contain summations Σ that extend over subsequent terms in the product.)

$$\begin{aligned}
\sum_{\bar{j}} A_{\bar{i}\bar{j}} &= \sum_{q_1 \cdots q_n} A_{i, q_1 \cdots q_n} \\
&= \prod_{l=1}^{n} \left(\sum_{q_l} B^{(l)}_{p_1 \cdots p_n q_1 \cdots q_l} \right) \\
&= \left[\prod_{l=1}^{n-1} \left(\sum_{q_l} B^{(l)}_{p_1 \cdots p_n q_1 \cdots q_l} \right) \right] \left(\sum_{q_n} B^{(n)}_{p_n q_1 \cdots q_n} = 1 \right) \\
&= \prod_{l=1}^{n-1} \left(\sum_{q_l} B^{(l)}_{p_1 \cdots p_n q_1 \cdots q_l} \right) \\
&= 1, \text{ by induction on } n. \qquad (64)
\end{aligned}$$

Likewise

$$\begin{aligned}
\sum_{\bar{i}} A_{\bar{i}\bar{j}} &= \sum_{p_1 \cdots p_n} A_{i, p_1 \cdots p_n} \\
&= \prod_{l=n}^{1} \left(\sum_{p_l} B^{(l)}_{p_1 \cdots p_n q_1 \cdots q_l} \right) \\
&= \left[\prod_{l=n}^{2} \left(\sum_{p_l} B^{(l)}_{p_1 \cdots p_n q_1 \cdots q_l} \right) \right] \left(\sum_{p_1} B^{(1)}_{p_1 \cdots p_n q_1} = 1 \right) \\
&= \prod_{l=n}^{2} \left(\sum_{p_l} B^{(l)}_{p_1 \cdots p_n q_1 \cdots q_l} \right) \\
&= 1, \text{ by induction.} \qquad (65)
\end{aligned}$$

B.2. Coarse Butterflies

A less restrictive form for A may be derived as follows. Let k be roughly $n/2$. Then

$$i_1 = p_1 \ldots p_k, \quad i_2 = p_{k+1} \ldots p_n,$$
$$j_1 = q_1 \ldots q_k, \quad j_2 = q_{k+1} \ldots q_n;$$

from eqn (63),

$$\begin{aligned}
A_{\bar{i}} &= \prod_{l=1}^{n} B^{(l)}_{p_1 \cdots p_n q_1 \cdots q_l} \\
&= \left(\prod_{l=1}^{k} B^{(l)}_{p_1 \cdots p_n q_1 \cdots q_l} \right) \left(\prod_{l=k+1}^{n} B^{(l)}_{p_1 \cdots p_n q_1 \cdots q_l} \right) \\
&= \left(\prod_{l=1}^{k} B^{(l)}_{p_1 \cdots p_n q_1 \cdots q_k} \right) \left(\prod_{l=k+1}^{n} B^{(l)}_{p_{k+1} \cdots p_n q_1 \cdots q_n} \right) \\
&= A^{(1)}_{i_1 j_2 \cdot n} A^{(2)}_{j_2 \cdot n i_2} \qquad (66)
\end{aligned}$$

and as in eqn (64) one can derive the constraints

$$\begin{aligned}
\sum_{j_1} A^{(1)}_{i_1 j_2 \cdot j_1} &= \prod_{l=1}^{k} \left(\sum_{q_l} B^{(l)}_{p_1 \cdots p_n q_1 \cdots q_l} \right) \\
&= \left[\prod_{l=1}^{k-1} \left(\sum_{q_l} B^{(l)}_{p_1 \cdots p_n q_1 \cdots q_l} \right) \right] \left(\sum_{q_k} B^{(k)}_{p_k \cdots p_n q_1 \cdots q_k} = 1 \right) \\
&= \prod_{l=1}^{k-1} \left(\sum_{q_l} B^{(l)}_{p_1 \cdots p_n q_1 \cdots q_l} \right) \\
&= 1, \text{ by induction on } k. \qquad (67)
\end{aligned}$$

Likewise

$$\begin{aligned}
\sum_{i_1} A^{(1)}_{i_1 j_2 \cdot j_1} &= \prod_{l=k}^{1} \left(\sum_{p_l} B^{(l)}_{p_1 \cdots p_n q_1 \cdots q_l} \right) \\
&= \left[\prod_{l=k}^{2} \left(\sum_{p_l} B^{(l)}_{p_1 \cdots p_n q_1 \cdots q_l} \right) \right] \left(\sum_{p_1} B^{(1)}_{p_1 \cdots p_n q_1} = 1 \right) \\
&= \prod_{l=k}^{2} \left(\sum_{p_l} B^{(l)}_{p_1 \cdots p_n q_1 \cdots q_l} \right) \\
&= 1, \text{ by induction.} \qquad (68)
\end{aligned}$$

The constraints on $A^{(2)}$ and $\bar{A}$ are similar. Thus the constraints on $B^{(l)}$ imply the less restrictive constraints on $A^{(1)}$ and $A^{(2)}$, which we then adopt as the only constraints operating on A.

This completes the proof that any permutation matrix M can be represented by $\mathcal{O}(N^{3/2})$ variables in the form of eqn (49) with constraints as in eqn (50).

APPENDIX C. FULL BUTTERFLY NEURAL NETS

The arguments of Appendix B can be generalized to yield a series of reduced objectives interpolating between $\mathcal{O}(N^{3/2})$ and $\mathcal{O}(N \log N)$ variables, each having about the same number of connections as variables. In Appendix B the idea was to express each index i in base $\sqrt{N} = 2^{n/2} = 2^k$, by dividing the indices $p_1 \ldots p_n$ of is binary expansion in two groups. We may instead divide the n binary indices into m groups of size k, with $n = km$, deriving the base 2^k expansion $i = i_1 \cdots i_m$. (We have dealt with the special cases $k = n$ and $k = n/2$.) Then

$$\begin{aligned}
A_{\bar{i}\bar{j}} &= \prod_{l=1}^{n} B^{(l)}_{p_1 \cdots p_n q_1 \cdots q_l} \\
&= \prod_{l=1}^{m} \left(\prod_{r=1}^{k} B^{(l,r)}_{p_{l,r} \cdots p_n q_1 \cdots q_{l,r}} \right) \\
&= \prod_{l=1}^{m} A^{(l)}_{i_l j_m \cdot j_1 i_{l-1}} \qquad (69)
\end{aligned}$$

as in eqn (66). The constraints on $A^{(l)}$ are, as usual,

$$\sum_{i_l} A^{(l)}_{i_l j_m \cdot j_1 i_{l-1}} = 1 = \sum_{j_l} A^{(l)}_{i_l j_m \cdot j_1 i_{l-1}} \tag{70}$$

which are generally less restrictive than the original constraints on the butterfly switches B.

From eqn (52), our problem is to reduce

$$E_1(A) = -c_1 \sum_{\bar{i}} A_{\bar{i}\bar{j}} x_i e_j \tag{71}$$

(where e_i is any expression) upon substituting eqn (69).

If we take $e_i = e^{(m)}_{i_1 \cdots i_m}$, $A_i = C^{(m)}_{i_1 \cdots i_m j_1 \cdots j_m}$, and $E_1(A) = E_1^{(m)}(C^{(m)})$, then we may use induction on a to reduce

$$E_1^{(a)}(C^{(a)}) = -c_1 \sum_{\substack{i_1 \cdots i_m \\ j_1 \cdots j_a}} C^{(a)}_{i_1 \cdots i_m j_1 \cdots j_a} x_{i_1 \cdots i_m} e^{(a)}_{i_{a+1} \cdots i_m j_1 \cdots j_a} \quad (a \geq 2), \tag{72}$$

where

$$C^{(a)}_{i_1 \cdots i_m j_1 \cdots j_a} \equiv \prod_{l=1}^{a} A^{(l)}_{i_{l-1} \cdots i_m j_1 \cdots j_l} \tag{73}$$

Then

$$C^{(a)}_{i_1 \cdots i_m j_1 \cdots j_a} = C^{(a-1)}_{i_1 \cdots i_m j_1 \cdots j_{a-1}} A^{(a)}_{i_{a-1} \cdots i_m j_1 \cdots j_a}$$

whence

$$\begin{aligned}
E_1^{(a)} = -\frac{c_1}{2} \sum_{\substack{i_a \cdots i_m \\ j_1 \cdots j_{a-1}}} \Bigg[& \Bigg(\sum_{i_1 \cdots i_{a-1}} C^{(a-1)}_{i_1 \cdots i_m j_1 \cdots j_{a-1}} x_{i_1 \cdots i_m} \\
& + \sum_{j_a} A^{(a)}_{i_a \cdots i_m j_1 \cdots j_a} e^{(a)}_{i_{a+1} \cdots i_m j_1 \cdots j_a} \Bigg)^2 \\
& - \Bigg(\sum_{i_1 \cdots i_{a-1}} C^{(a-1)}_{i_1 \cdots i_m j_1 \cdots j_{a-1}} x_{i_1 \cdots i_m} \Bigg)^2 \\
& - \Bigg(\sum_{j_a} A^{(a)}_{i_a \cdots i_m j_1 \cdots j_a} e^{(a)}_{i_{a+1} \cdots i_m j_1 \cdots j_a} \Bigg)^2 \Bigg] \\
\longrightarrow -c_1 \Bigg[& \sum_{i_1 \cdots i_m} \sum_{j_1 \cdots j_{a-1}} C^{(a-1)}_{i_1 \cdots i_m j_1 \cdots j_{a-1}} x_{i_1 \cdots i_m} \\
& \times (\sigma^{(a-1)}_{i_a \cdots i_m j_1 \cdots j_{a-1}} - \tau^{(a-1)}_{i_a \cdots i_m j_1 \cdots j_{a-1}}) \\
& + \sum_{i_a \cdots i_m} \sum_{j_1 \cdots j_a} A^{(a)}_{i_a \cdots i_m j_1 \cdots j_a} e^{(a)}_{i_{a+1} \cdots i_m j_1 \cdots j_a} \\
& \times (\sigma^{(a-1)}_{i_a \cdots i_m j_1 \cdots j_{a-1}} - \omega^{(a-1)}_{i_a \cdots i_m j_1 \cdots j_{a-1}}) \\
& + \tfrac{1}{2} \sum_{i_a \cdots i_m} \sum_{j_1 \cdots j_{a-1}} [-(\sigma^{(a-1)}_{i_a \cdots i_m j_1 \cdots j_{a-1}})^2 \\
& + (\tau^{(a-1)}_{i_a \cdots i_m j_1 \cdots j_{a-1}})^2 + (\omega^{(a-1)}_{i_a \cdots i_m j_1 \cdots j_{a-1}})^2] \Bigg].
\end{aligned} \tag{74}$$

Note that, upon identifying

$$\sigma^{(a-1)} \equiv \tau^{(a-1)} \equiv e^{(a-1)},$$

we have an induction step $a \to a - 1$, $\forall a \geq 2$, which decreases the number of neurons and connections in the network. By induction on a, one may reduce eqn (52) to:

$$\begin{aligned}
E_{\text{sort}} = -c_1 \Bigg[& \sum_{i_1 \cdots i_m j_1} A^{(1)}_{i_1 \cdots i_m j_1} x_{i_1 \cdots i_m} \\
& \times (\sigma^{(1)}_{i_2 \cdots i_m j_1} - \tau^{(1)}_{i_2 \cdots i_m j_1}) \\
& + \sum_{a=2}^{m-1} \sum_{\substack{i_a \cdots i_m \\ j_1 \cdots j_a}} A^{(a)}_{i_a \cdots i_m j_1 \cdots j_a} \\
& \times (\sigma^{(a-1)}_{i_a \cdots i_m j_1 \cdots j_{a-1}} - \omega^{(a-1)}_{i_a \cdots i_m j_1 \cdots j_{a-1}}) \\
& \times (\sigma^{(a)}_{i_{a+1} \cdots i_m j_1 \cdots j_a} - \tau^{(a)}_{i_{a+1} \cdots i_m j_1 \cdots j_a}) \\
& + \sum_{i_m j_1 \cdots i_m} A^{(m)}_{i_m j_1 \cdots i_m} \\
& \times (\sigma^{(m-1)}_{i_m j_1 \cdots i_{m-1}} - \omega^{(m-1)}_{i_m j_1 \cdots i_{m-1}})(a_{j_1 \cdots i_m} - b_{j_1 \cdots i_m}) \\
& + \tfrac{1}{2} \sum_{a=1}^{m-1} \sum_{\substack{i_{a+1} \cdots i_m \\ j_1 \cdots j_a}} [-(\sigma^{(a)}_{i_{a+1} \cdots i_m j_1 \cdots j_a})^2 \\
& + (\tau^{(a)}_{i_{a+1} \cdots i_m j_1 \cdots j_a})^2 + (\omega^{(a)}_{i_{a+1} \cdots i_m j_1 \cdots j_a})^2] \\
& + \left\{ \text{same, with } \begin{matrix} x \longrightarrow y & \sigma \longrightarrow \hat{\sigma} \\ B \longrightarrow \hat{B} & \tau \longrightarrow \hat{\tau} \\ b \longrightarrow \hat{b} & \omega \longrightarrow \hat{\omega} \\ a \longrightarrow a & \end{matrix} \right\} \\
& + \tfrac{1}{2} \sum_{i_1 \cdots i_m} [-a^2_{i_1 \cdots i_m} + b^2_{i_1 \cdots i_m} + \hat{b}^2_{i_1 \cdots i_m}] \Bigg].
\end{aligned} \tag{75}$$

The number of variables and connections for this objective is $\mathcal{O}(mN^{1+1/m})$, $1 \leq m \leq n$, which takes on values N^2, $2N^{3/2}$, $3N^{4/3}$, . . . $N \log N$.

C Appendix: Bayesian Inference on Visual Grammars by Neural Nets that Optimize

[Shortened version accepted, subject to revisions, for IEEE Transactions on Neura

Bayesian Inference on Visual Grammars by Neural Nets that Optii

Eric Mjolsness
Department of Computer Science
Yale University

May 1, 199

YALEU-DCS-TR-854

Abstract

We exhibit a systematic way to derive neural *nets for vision problems*. It involves formulatin a vision problem as Bayesian inference or decision on a comprehensive model of the visual domai given by a probabilistic *grammar*. A key feature of this grammar is the way in which it eliminate *model information*, such as object labels, as it produces an image; *correspondance problems* and othe noise removal tasks result. The neural nets that arise most directly are generalized assignment net works. Also there are transformations which naturally yield improved algorithms such as correlatio matching in scale space and the Frameville neural nets for high-level vision. Networks derived th way generally have objective functions with spurious local minima; such minima may commonly L avoided by dynamics that *include deterministic annealing*, for example recent improvements to Mea Field Theory dynamics. The grammatical method of neural net design allows domain knowledge t enter from all levels of the grammar, including "abstract" levels remote from the final image dat. *and may permit new kinds of learning* as well.

Contents

1 INTRODUCTION

We show how to derive various optimizing neural networks, which represent quantitative visual models and match them to data, from fundamental considerations of Bayesian reasoning. Doing so illustrates a design methodology which starts from first principles, namely a probabilistic model of a visual domain, and proceeds to a neural network which performs visual tasks. The key problem is to choose probability distributions sufficiently intricate to model general visual tasks and yet tractable enough for theory. We do this by means of probabilistic and expressive grammars which model the image-formation process, including heterogeneous sources of noise each modelled with a grammar rule. In particular these grammars include a crucial "renumbering" rule that removes the undetectable internal labels (or indices) of detectable features and substitutes an uninformed numbering scheme used by the perceiver.

For such grammars, in which every rule has a simple Boltzmann probability distribution, it is straightforward to generate a neural network as follows: (1) Obtain the grammar, by detailed modelling or by automated learning from examples. (2) Compute the joint Boltzmann probability distribution on images (or pictures) and their grammatical explanations. (3) Express desired averages under this distribution in terms of the optimization of an objective function E. This step usually employs Mean Field Theory approximations; the scaling properties and practicality of such approximations have been greatly improved by (Simic, 1990b; Peterson and Soderberg, 1989; Van den Bout and Miller, III, 1990) for matching problems similar to those we encounter. (4) Introduce optimizing neural net dynamics for E.

The procedure becomes more elaborate and malleable by using valid *transformations* (e.g. of probability distributions or objective functions) at each step to reduce network cost, improve network performance or achieve network implementability in some technology. A small catalog of valid objective function transformations for neural nets is presented in (Mjolsness and Garrett, 1990), and the present paper illustrates several transformations of probability distributions. The entire method is sketched in Figure 1.

This paper is organized as follows. In Section 2 we introduce an example grammar which models a simple picture-formation process; we derive a joint probability distribution on models and images (or pictures) and discuss various questions which could be answered by computing moments of this distribution; we derive neural nets for doing such computations, both with and without using "match neurons" which explicitly hypothesize correspondance between model and data objects. The network without such neurons is interpreted as the relatively efficient algorithm of correlation matching in scale space. In Section 3 we demonstrate such nets on an image registration problem. In Section

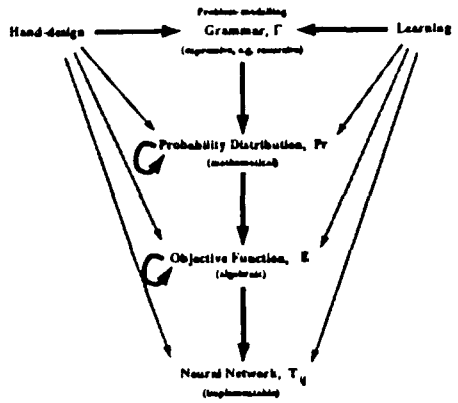


Figure 1: A neural network design methodology. Solid arrows constitute the recommended procedure. The arrow from Pr to E may be realized by approximations from statistical physics, such as Mean Field Theory. Circular arrows represent valid transformations, such as fixed-point preserving transformations of objective functions.

4 we exhibit a variety of probabilistic grammars and the joint probability distributions they imply, including a network for recognizing simple flexible objects. There we also derive the previously studied "Frameville" neural networks for high-level vision from a simple type of grammar. It results from "pushing" the renumbering operation back to more and more abstract *levels of the grammar*, thereby breaking up a massive correspondance problem into many small correspondance and grouping problems that interact. To put the latter result in context, it suggests a natural way to derive neural nets that are capable of expressing abstractions usually reserved for symbolic computing, and indicates how symbolic computing in perceptual domains could be improved by derivation from underlying physical models akin to those used in physically-based computer graphics. In Section 5 we speculate on learning algorithms for the types of neural nets we are discussing, illustrated with an example involving inexact graph matching. In Section 6 we discuss the results and conclude.

2 EXAMPLE: A RANDOM-DOT GRAMMAR

The first example grammar is a generative model of pictures consisting of a number of dots (e.g. a : of delta functions) whose relative locations are determined by one out of M stored models. But dots are subject to unknown independent jitter and an unknown global translation, and the identi of the dots (their numerical labels) are hidden from the perceiver by a random permutation operat For example each model might represent an imaginary asterism of equally bright stars whose locat. have been corrupted by instrument noise. One useful task would be to recognize which model gener. the image data.

2.1 The Grammar

The random-dot grammar is shown below.

model and its location	Γ^0 : root	$\rightarrow$	instance of model α at $\mathbf{x}$
	$E_0(\mathbf{x})$	$=$	$\frac{1}{2\sigma_x^2}\lvert\mathbf{x}\rvert^2$
dot locations	Γ^1 : instance$(\alpha, \mathbf{x})$	$\rightarrow$	$\{\text{dotloc}(\alpha, m, \dot{\mathbf{x}}_m = \mathbf{x} + \mathbf{u}_m^\alpha)\}$
	$E_1(\{\dot{\mathbf{x}}_m\})$	$=$	$-\log\prod_m \delta(\dot{\mathbf{x}}_m - \mathbf{x} - \mathbf{u}_m^\alpha)$, where $< \mathbf{u}_m^\alpha >_m = 0$
		$\approx$	$\lim_{\sigma_j \to 0} \frac{1}{2\sigma_j^2}\sum_m \lvert\dot{\mathbf{x}}_m - \mathbf{x} - \mathbf{u}_m^\alpha\rvert^2 + c(\sigma_j)$
dot jitter	Γ^2 : dotloc$(\alpha, m, \dot{\mathbf{x}}_m)$	$\rightarrow$	dot$(m, \mathbf{x}_m)$
	$E_2(\mathbf{x}_m)$	$=$	$\frac{1}{2\sigma_{jt}^2}\lvert\dot{\mathbf{x}}_m - \mathbf{x}_m\rvert^2$
choose random permutation	Γ^3 : $\{\text{dot}(m, \mathbf{x}_m)\}$	$\rightarrow$	$\{\text{dot}(m, \mathbf{x}_m), P_{m,t}\}$
	$E_3(\{P_{m,t}\})$	$=$	$-\log \Pr(P)$, where P is a permutation
renumber all dots	Γ^4 : $\{\text{dot}(m, \mathbf{x}_m), P_{m,t}\}$	$\rightarrow$	$\{\text{imagedot}(\mathbf{x}_t = \sum_m P_{m,t}\mathbf{x}_m)\}$
	$E_4(\{\mathbf{x}_t\})$	$=$	$-\log\prod_t \delta(\mathbf{x}_t - \sum_m P_{m,t}\mathbf{x}_m)$

The operation of this grammar is illustrated in Figure 2. We will show that this grammar is equiv. to a grammar with fewer rules:

model and location	Γ^0 : root	$\rightarrow$	instance of model α at $\mathbf{x}$
	$E_0(\mathbf{x})$	$=$	$\frac{1}{2\sigma_x^2}\lvert\mathbf{x}\rvert^2$
jittered dot locations	Γ^1 : instance$(\alpha, \mathbf{x})$	$\rightarrow$	$\{\text{dot}(\alpha, m, \mathbf{x}_m)\}$
	$E_1(\{\mathbf{x}_m\})$	$=$	$\frac{1}{2\sigma_{jt}^2}\sum_m(\mathbf{x}_m - \mathbf{x} - \mathbf{u}_m^\alpha)^2$, where $< \mathbf{u}_m^\alpha >_m = 0$
		$\approx$	$\lim_{\sigma_s \to 0} \frac{1}{2(\sigma_{jt}^2 + \sigma_s^2)}\sum_m \lvert\mathbf{x}_m - \mathbf{x} - \mathbf{u}_m^\alpha\rvert^2 + c(\sigma_s)$
scramble all dots	Γ^2 : $\{\text{dot}(m, \mathbf{x}_m)\}$	$\rightarrow$	$\{\text{imagedot}(\mathbf{x}_t = \sum_m P_{m,t}\mathbf{x}_m)\}$
	$E_2(\{\mathbf{x}_t\})$	$=$	$-\log\lvert\Pr(P)\prod_t \delta(\mathbf{x}_t - \sum_m P_{m,t}\mathbf{x}_m)\rvert$ where P is a permutation

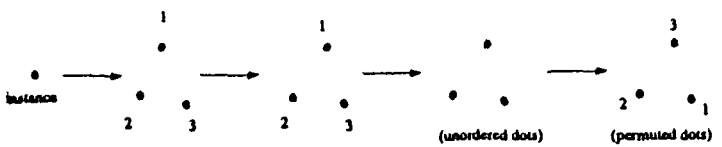


Figure 2: Operation of random dot grammar. The first arrow illustrates dot placement; the next shows dot jitter; the next arrow shows the pure, un-numbered feature locations; and the final arrow is the uninformed renumbering scheme of the perceiver.

The generative process specified by this grammar is to start with nothing (the "root" of a parse tree) and generate one instance of a model chosen randomly from a list of known models. Let the chosen model number be α. Rule Γ^0 also places the instance on the image plane with a Gaussian distribution of locations $\mathbf{x}$. Given such an instance, the only applicable rule is Γ^1 which replaces it with a set of dots whose locations $\mathbf{x}_m$ are Gaussian-distributed displacements of ideal locations given by $\mathbf{x} + \mathbf{u}^\alpha_m$. The final rule is special: its input is the set of all dots generated by the grammar, and it replaces them with a permuted set of image dots at the same set of locations. In other words it relabels the dots from index m to index i by means of a permutation P_{mi}. The permutation probability distribution $\Pr(P)$ will be specified in the next section. We will show that it is indistinguishable from the uniform distribution on permutations.

2.2 Final Probability Distribution

The probability distribution associated with a particular rule Γ^r is

$$\Pr(\text{new terms}, \{\text{new parameters}\} | \text{old terms}, \{\text{old parameters}\}) = e^{-\beta E_r}/Z_r \tag{3}$$

where $\beta \to 1$. Such conditional probabilities can be repeatedly combined in the usual way:

$$\Pr(\xi, x) = \Pr(x|\xi)\Pr(\xi) \tag{4}$$

to yield a final joint probability distribution for the entire grammar. For Grammar 1, we could multiply to get an expression for $\Pr(\alpha, \mathbf{x}, \{\dot{\mathbf{x}}_m\}, \{\mathbf{x}_m\}, \{P_{m,i}\}, \{\mathbf{x}_i\})$. However we are usually interested in computing some average in this distribution, i.e. some moment of this function.

To begin with, the delta functions in rules 1 and 4 completely determine $\dot{\mathbf{x}}_m$ and $\mathbf{x}_m$, respectively, in terms of the other variables; so these are usually integrated out of any interesting average. That

leaves $\Pr(\alpha, \mathbf{x}, \{P_{m,i}\}, \{\mathbf{x}_i\})$. For this reason Grammar 2 is obtained by combining Grammar 1's rules and 2 (integrating out $\dot{\mathbf{x}}_m$) and rules 3 and 4 (leaving in $\mathbf{x}_m$). In Section 2.3 we will show that the P variables can often be summed out without losing the answer to real problems. As an example, let us calculate $\Pr(\alpha, \mathbf{x}, \{\mathbf{x}_i\})$ for this grammar.

Calculating the joint probability distribution is especially easy for this grammar because the grammar rules are not recursive. The grammar just consists of a sequence of three stages corresponding to rules 0-2. After rule 0,

$$\Pr^0(\alpha, \mathbf{x}) = \frac{1}{A}\left(\frac{1}{\sqrt{2\pi}\sigma_r}\right)^2 e^{-\frac{1}{2\sigma_r^2}|\mathbf{x}|^2} \tag{5}$$

where A is the number of models to choose from. The probability after rule 1 is

$$\begin{aligned}\Pr^1(\alpha, \mathbf{x}, \{\mathbf{x}_m\}) &= \Pr^1(\{\mathbf{x}_m\}|\alpha, \mathbf{x})\Pr^0(\alpha, \mathbf{x}) \\ &= \frac{1}{A}\left(\frac{1}{\sqrt{2\pi}\sigma_r}\right)^2 \left(\frac{1}{\sqrt{2\pi}\sigma_{jt}}\right)^{2N} e^{-\left(\frac{1}{2\sigma_r^2}|\mathbf{x}|^2 + \frac{1}{2\sigma_{jt}^2}\sum_m |\mathbf{x}_m - \mathbf{x} - \mathbf{u}^\alpha_m|^2\right)}.\end{aligned} \tag{6}$$

To finish the calculation we must consider $\Pr(P)$. This is the probability of a given renumbering from object-generation indices m to image indices i. This part of the grammar models the fact that the *object-generation indices* m are generally inaccessable to the perceiver, though if they were known the perception problem would be nearly solved. One model for P is to feign ignorance of the permutation and use the maximum entropy distribution on P, namely a uniform distribution. This model seems artificial because there is no actual uniform-probability scrambling mechanism in natural image-generation processes.

Alternatively we could model the renumbering process as deterministically *sorting* the raw dot position data $\mathbf{x}_m$ according to some scalar criterion $f(\mathbf{x}_m)$ so that low values of f are indexed by integers i with low values of some set of weights w_i (e.g. $w_i = i$), and high f's correspond to high w_i. This sorting is done by whatever part of the perceiver turns images into some representation of low-level symbols. For example, f could encode the lexicographic ordering of a 2-d array of pixels given by the raster or scan-line sequence, and the dots would be indexed in that order. Or f could order the 2-d array of pixels according to the discretization of any space-filling curve. It is important that f does not depend on quantities like m or α that require perception to deduce. $f(\mathbf{x})$ should also be unique, i.e. f is an invertable function. Then, $\Pr(P)$ is

$$\Pr(P) = \lim_{\beta_{sort} \to \infty} e^{\beta_{sort}\sum_{m,i} P_{m,i} f(\mathbf{x}_m) w_i}/Z_P \tag{7}$$

and E_2 contributes

$$\begin{aligned}
\sum_{(P)} e^{-E_2(\mathbf{x}_i)} &= \lim_{\beta_{sort}\to\infty} \sum_{\left\{ \begin{smallmatrix} P \mid P \text{ is a} \\ \text{permutation} \end{smallmatrix} \right\}} \frac{e^{\beta_{sort}\sum_m P_{m,i} f(\mathbf{x}_m)\omega_i}}{Z_P} \prod_i \delta(\mathbf{x}_i - \textstyle\sum_m P_{m,i}\mathbf{x}_m) \\
&= \lim_{\beta_{sort}\to\infty} \sum_{\left\{ \begin{smallmatrix} P \mid P \text{ is a} \\ \text{permutation} \end{smallmatrix} \right\}} \frac{e^{\beta_{sort}\sum_i f(\sum_m P_{m,i}\mathbf{x}_m)\omega_i}}{Z_P} \prod_i \delta(\mathbf{x}_i - \textstyle\sum_m P_{m,i}\mathbf{x}_m) \\
&= \lim_{\beta_{sort}\to\infty} \sum_{\left\{ \begin{smallmatrix} P \mid P \text{ is a} \\ \text{permutation} \end{smallmatrix} \right\}} \frac{e^{\beta_{sort}\sum_i f(\mathbf{x}_i)\omega_i}}{Z_P} \prod_i \delta(\mathbf{x}_i - \textstyle\sum_m P_{m,i}\mathbf{x}_m) \qquad (8) \\
&= \left(\lim_{\beta_{sort}\to\infty} e^{\beta_{sort}\sum_i f(\mathbf{x}_i)\omega_i} / \sum_{\left\{ \begin{smallmatrix} P \mid P \text{ is a} \\ \text{permutation} \end{smallmatrix} \right\}} e^{\beta_{sort}\sum_m P_{m,i} f(\mathbf{x}_m)\omega_i} \right) \\
&\quad \times \sum_{\left\{ \begin{smallmatrix} P \mid P \text{ is a} \\ \text{permutation} \end{smallmatrix} \right\}} \prod_i \delta(\mathbf{x}_i - \textstyle\sum_m P_{m,i}\mathbf{x}_m).
\end{aligned}$$

Here the second term enforces the constraint that $\mathbf{x}_i$ be a permutation of $\mathbf{x}_m$, and in this circumstance the first term constrains $\mathbf{x}_i$ to be in order according to f. Thus we may simplify the first term when it is multiplied by the second:

$$\sum_{(P)} e^{-E_2(\mathbf{x}_i)} = \Theta(\{\mathbf{x}_i\}) \sum_{\left\{ \begin{smallmatrix} P \mid P \text{ is a} \\ \text{permutation} \end{smallmatrix} \right\}} \prod_i \delta(\mathbf{x}_i - \sum_m P_{m,i}\mathbf{x}_m), \qquad (9)$$

where $\Theta(\{\mathbf{x}_i\})$ is 1 if $\{\mathbf{x}_i\}$ are in order according to f, and zero otherwise. In the first step of this derivation we have used a crucial property of permutation matrices: that $\sum_m P_{m,i} f(\mathbf{x}_m) = f(\sum_m P_{m,i}\mathbf{x}_m)$ because exactly one element of $\{P_{m,i} | m \in \{1 \ldots N\}\}$ is nonzero.

Since $\Theta(\{\mathbf{x}_i\})$ does not depend on $\{\mathbf{x}_m\}$, it will cancel out when we compute conditional probabilities such as $\Pr^{\text{final}}(\alpha, \mathbf{x} | \{\mathbf{x}_i\})$ (c.f. equation (13) below.) In other words, the sorting model of $\Pr(P)$ is equivalent to the uniform-distribution model as far as E_2 and hence $\Pr^2$ are concerned.

So,

$$\begin{aligned}
\Pr^{\text{final}}(\alpha, \mathbf{x}, \{\mathbf{x}_i\}) &= \frac{\Theta(\{\mathbf{x}_i\})}{A} \left(\frac{1}{\sqrt{2\pi}\sigma_r}\right)^2 \left(\frac{1}{\sqrt{2\pi}\sigma_{jt}}\right)^{2N} \sum_{\left\{ \begin{smallmatrix} P \mid P \text{ is a} \\ \text{permutation} \end{smallmatrix} \right\}} \\
&\quad \int d\{\mathbf{x}_m\} \prod_i \delta(\mathbf{x}_i - \sum_m P_{m,i}\mathbf{x}_m) e^{-\left(\frac{1}{2\sigma_r^2}|\mathbf{x}|^2 + \frac{1}{2\sigma_{jt}^2}\sum_m |\mathbf{x}_m - \mathbf{x} - \mathbf{u}_m^\alpha|^2\right)} \\
&= \frac{\Theta(\{\mathbf{x}_i\})}{A} \left(\frac{1}{\sqrt{2\pi}\sigma_r}\right)^2 \left(\frac{1}{\sqrt{2\pi}\sigma_{jt}}\right)^{2N} \qquad (10) \\
&\quad \times \sum_{\left\{ \begin{smallmatrix} P \mid P \text{ is a} \\ \text{permutation} \end{smallmatrix} \right\}} e^{-\left(\frac{1}{2\sigma_r^2}|\mathbf{x}|^2 + \frac{1}{2\sigma_{jt}^2}\sum_m |\sum_i P_{m,i}\mathbf{x}_i - \mathbf{x} - \mathbf{u}_m^\alpha|^2\right)}
\end{aligned}$$

and finally

$$\begin{aligned}
\Pr^{\text{final}}(\alpha, \mathbf{x}, \{\mathbf{x}_i\}) &= \frac{\Theta(\{\mathbf{x}_i\})}{A} \left(\frac{1}{\sqrt{2\pi}\sigma_r}\right)^2 \left(\frac{1}{\sqrt{2\pi}\sigma_{jt}}\right)^{2N} \\
&\quad \times \sum_{\left\{ \begin{smallmatrix} P \mid P \text{ is a} \\ \text{permutation} \end{smallmatrix} \right\}} e^{-\beta \sum_{m,i} P_{m,i}\left(\frac{1}{2N\sigma_r^2}|\mathbf{x}|^2 + \frac{1}{2\sigma_{jt}^2}|\mathbf{x}_i - \mathbf{x} - \mathbf{u}_m^\alpha|^2\right)} \qquad (11)
\end{aligned}$$

The inverse temperature β just introduced must of course be set to one. But for Bayesian inference algorithms this may be done by gradually increasing β from zero, a procedure called "annealing", which often has the effect of avoiding local minima during a computation.

In Section 2.4 we review how the Boltzmann distribution (11) and its derivatives may be approximated by a neural net involving quadratic match neurons related to $P_{m,i}$. In Section 2.5 we exhibit a new way to derive even simpler, though sometimes less accurate, neural nets for such problems by eliminating the P variables.

Equation (11) is representative of most of the grammatical probability distributions we will derive in one important way: it is a Boltzmann distribution whose objective function is a generalized "assignment" objective function. The "assignment problem" (Bertsekas and Tsitsiklis, 1989) is to minimize $E = \sum_{\alpha\alpha} P_{\alpha\alpha} W_{\alpha\alpha}$ over permutations P, for constant weights $W \geq 0$. A neural net approach to this problem is analysed in (Kosowsky and Yuille, 1991). In equation (11) the assignment problem objective is generalized because the weights W are now functions of real-valued parameters, as will generally be the

case for grammatical probability distributions:

$$E_{\text{final}}(\alpha, P, \mathbf{x}) = \sum_{mi} P_{m,i} \left(\frac{1}{2N\sigma_e^2} |\mathbf{x}|^2 + \frac{1}{2\sigma_{ji}^2} |\mathbf{x}_i - \mathbf{x} - \mathbf{u}_m^\alpha|^2 \right) \tag{12}$$

The sum over permutations may be approximated by an optimization over near-permutations, as we will see, and the fact that P appears only linearly in E_{final} makes such optimization problems easier.

2.3 Inference and Decision Problems

We now show how to pose a variety of problems which could arise in situations modeled by the grammer and whose solution can be expressed as sums over the P configuration space as in the previous section.

A simple recognition problem might involve looking at data $\{\mathbf{x}_i\}$ and inferring the most likely model (α) and its position ($\mathbf{x}$). We must find

$$\begin{aligned} \text{argmax}_{\alpha,\mathbf{x}} \Pr(\alpha, \mathbf{x} | \{\mathbf{x}_i\}) &= \text{argmax}_{\alpha,\mathbf{x}} \frac{\Pr(\alpha, \mathbf{x}, \{\mathbf{x}_i\})}{\Pr(\{\mathbf{x}_i\})} \quad \text{(Bayesian inference)} \\ &= \text{argmax}_{\alpha,\mathbf{x}} \Pr(\alpha, \mathbf{x}, \{\mathbf{x}_i\}) \\ &= \text{argmax}_{\alpha,\mathbf{x}} \sum_{\left\{ \begin{smallmatrix} P | \ P \text{ is a} \\ \text{permutation} \end{smallmatrix} \right\}} \Pr(\alpha, \mathbf{x}, \{P_{m,i}\}, \{\mathbf{x}_i\}). \end{aligned} \tag{13}$$

Note that the combination of equations (13) and (4) perform Bayesian inference: they determine *Pr(model params|data) in terms of* forward conditional probabilities including Pr(data|model params). From equation 13, this recognition problem involves computing the function $\Pr(\alpha, \mathbf{x}, \{\mathbf{x}_i\})$ for which we must integrate (sum) out the P variables.

Other problems might require the computation of the average $\langle P_{m,i} \rangle$, which still requires summing over the P configurations. Indeed most if not all inference problems for the grammar can be formulated in terms of such sums. A Mean Field Theory approximation to these high-dimensional sums may be performed by a neural net (Hopfield, 1984; Peterson and Soderberg, 1989; Simic, 1990b; Yuille, 1990; Simic, 1990a; Tresp, 1991), as we will review in Section 2.4.

An even simpler recognition problem would be to infer the most likely model α from data $\{\mathbf{x}_i\}$ without regard to its position. This is just like the previous problem except we want to integrate out $\mathbf{x}$. The integral with respect to arbitrary translations $\mathbf{x}$ could be done analytically, but corresponding integrals over 2-d rotation or other Lie group distortions that could be added to the grammar (see Section 4.1) must be approximated and we could treat translations the same way. In particular, a

saddle point approximation to the integral over $\mathbf{x}$ (involved in the MFT method) requires finding the most likely value of $\mathbf{x}$ but not reporting it as part of the answer.

Here are two more examples of problems posed in terms of the grammar which could be solved by summing over all P configurations. One might want to find the single configuration of $\alpha, \mathbf{x}, \{\mathbf{x}_m\}$, and $\{P_{m,i}\}$ which is most probable given observations $\{\mathbf{x}_i\}$, i.e. the *maximum a posteriori* (MAP) estimate of all these variables. We may compute the total partition function

$$\begin{aligned} Z(\beta, \lambda, \mu, \{\nu\}, \{\pi\}) &= \sum_\alpha \int d\mathbf{x} \int d\{\mathbf{x}_m\} \sum_{\left\{ \begin{smallmatrix} P | \ P \text{ is a} \\ \text{permutation} \end{smallmatrix} \right\}} \\ &\exp \left[-\beta E_{\text{final}} + \lambda\alpha + \mu \cdot \mathbf{x} + \sum_m \nu_m \cdot \mathbf{x}_m + \sum_{m,i} \pi_{m,i} P_{m,i} \right]. \end{aligned} \tag{14}$$

Then, average values of P and the other variables are derivatives of $\log Z$ evaluated at $\lambda, \mu, \nu, \pi = 0$. As $\beta \to \infty$, these averages approach the MAP configuration if it is unique. Note the departure from $\beta \to 1$ which would be used for the other problems discussed in this section. In fact the argmax taken in equation (13) could also be done by a zero-temperature limit, but that would involve a *different* β *parameter* than the one used in integrating out P.

A second type of problem involves *deciding what to do* based on $\Pr(\alpha, \mathbf{x} | \{\mathbf{x}_i\})$. For example, each model α might have a known interesting part whose position relative to the object center of mass is $\mathbf{c}^\alpha$. The problem is to set the variable $\mathbf{y}$ near this location (e.g. to point a telescope), under uncertainty in the values of α and $\mathbf{x}$. This could be formalized as maximizing the probable reward

$$\begin{aligned} F(\mathbf{y} | \{\mathbf{x}_i\}) &= \sum_\alpha \int d\mathbf{x} e^{-\frac{1}{2\sigma_y^2}(\mathbf{x} + \mathbf{c}^\alpha - \mathbf{y})^2} \Pr(\alpha, \mathbf{x} | \{\mathbf{x}_i\}) \\ &\propto \sum_\alpha \int d\mathbf{x} \sum_{\left\{ \begin{smallmatrix} P | \ P \text{ is a} \\ \text{permutation} \end{smallmatrix} \right\}} \exp -\beta \left[E_{\text{final}}(\alpha, P, \mathbf{x}) - \frac{1}{2\sigma_y^2}(\mathbf{x} + \mathbf{c}^\alpha - \mathbf{y})^2 \right] \end{aligned} \tag{15}$$

with respect to $\mathbf{y}$. Again we use Bayesian inference to get $\Pr(\alpha, \mathbf{x} | \{\mathbf{x}_i\})$, and again a sum over configurations of P is involved. The integral over $\mathbf{x}$ may be done analytically or approximately, as previously discussed.

Thus a wide variety of Bayesian inference and probabilistic decision problems may be reduced to calculating moments of $\Pr(\alpha, \mathbf{x}, \{P_{m,i}\}, \{\mathbf{x}_i\})$ that involve summing out the P configurations and integrating or maximizing with respect to the other random variables.

2.4 Neural Network with Match Variables

We review how configuration-space sums over P (along with other variables) may be approximated by quadratic match neural nets. For example we may compute $\Pr^J(\alpha, \mathbf{x}|\{\mathbf{x}_i\})$ as follows:

$$\begin{aligned}
\Pr^J(\alpha, \mathbf{x}|\{\mathbf{x}_i\}) &= C \sum_{\left\{ \begin{array}{l} \forall i \sum_m P_{m,i} = 1 \\ \wedge \sum_m P_{m,j} = 1 \end{array} \right\}} \exp - \sum_{mi} P_{m,i} \left(\frac{1}{2N\sigma_r^2} |\mathbf{x}|^2 + \frac{1}{2\sigma_{jt}^2} |\mathbf{x}_i - \mathbf{x} - \mathbf{u}_m^\alpha|^2 \right) \\
&= C \sum_{\left\{ \forall i \sum_m P_{m,i} = 1 \right\}} \prod_i \delta^K(\sum_m P_{m,i}, 1) \int_{-\infty}^{+\infty} d\{V_{mi}\} \prod_{m,i} \delta(V_{m,i} - P_{m,i}) \\
&\quad \exp - \textstyle\sum_{mi} V_{m,i} \left(\frac{1}{2N\sigma_r^2} |\mathbf{x}|^2 + \frac{1}{2\sigma_{jt}^2} |\mathbf{x}_i - \mathbf{x} - \mathbf{u}_m^\alpha|^2 \right)
\end{aligned} \tag{16}$$

where $\delta^K(n, m)$ is the Kronecker delta function on integers and $\delta(x)$ is the Dirac delta function on real numbers. Both have Gaussian representations, but we'll use an integral representation of the Dirac delta and the Gaussian representation $\delta^K(n, m) = \lim_{A \to \infty} \exp -(A/2)(n - m)^2$. Continuing,

$$\begin{aligned}
\Pr^J(\alpha, \mathbf{x}|\{\mathbf{x}_i\}) &= C \int_{-\infty}^{+\infty} d\{V_{mi}\} \sum_{\left\{ \forall i \sum_m P_{m,i} = 1 \right\}} \lim_{A \to \infty} \exp \left[-(A/2) \sum_i (\sum_m P_{m,i} - 1)^2 \right] \\
&\quad \int_{-i\infty}^{+i\infty} d\{U_{mi}\} e^{-\sum_{m,i} U_{m,i}(V_{m,i} - P_{m,i})} \exp -\beta \sum_{mi} V_{m,i} \left(\frac{1}{2N\sigma_r^2} |\mathbf{x}|^2 + \frac{1}{2\sigma_{jt}^2} |\mathbf{x}_i - \mathbf{x} - \mathbf{u}_m^\alpha|^2 \right) \\
&= C \lim_{A \to \infty} \int_{-\infty}^{+\infty} d\{V_{mi}\} \int_{-i\infty}^{+i\infty} d\{U_{mi}\} \exp \left[-\beta \sum_{mi} V_{m,i} \left(\frac{1}{2N\sigma_r^2} |\mathbf{x}|^2 + \frac{1}{2\sigma_{jt}^2} |\mathbf{x}_i - \mathbf{x} - \mathbf{u}_m^\alpha|^2 \right) \right] \\
&\quad \exp \left[-\beta(A/2) \sum_i (\sum_m V_{m,i} - 1)^2 - \sum_{m,i} U_{m,i} V_{m,i} \right] \sum_{\left\{ \forall i \sum_m P_{m,i} = 1 \right\}} \exp \sum_{m,i} U_{m,i} P_{m,i} \\
&= C \lim_{A \to \infty} \int_{-\infty}^{+\infty} d\{V_{mi}\} \int_{-i\infty}^{+i\infty} d\{U_{mi}\} \exp -\beta E_{eff}(\alpha, \mathbf{x}, \{U\}, \{V\})
\end{aligned} \tag{17}$$

where

$$\begin{aligned}
E_{eff}(\alpha, \mathbf{x}, \{U\}, \{V\}) \equiv & \textstyle\sum_{mi} V_{m,i} \left(\frac{1}{2N\sigma_r^2} |\mathbf{x}|^2 + \frac{1}{2\sigma_{jt}^2} |\mathbf{x}_i - \mathbf{x} - \mathbf{u}_m^\alpha|^2 \right) + (A/2) \textstyle\sum_i (\sum_m V_{m,i} - 1)^2 \\
& + (1/\beta) \sum_{m,i} U_{m,i} V_{m,i} - (1/\beta) \sum_m \log \left(\sum_i \exp U_{m,i} \right).
\end{aligned} \tag{18}$$

Up to this point the expression is exact; no approximations have been made. Now we approximate both the U and V integrals by way of the saddle points $(\{U^*\}, \{V^*\})$ of the objective function E_{eff}:

$$\begin{aligned}
\operatorname{argmax}_{\alpha, \mathbf{x}} \Pr^J(\alpha, \mathbf{x}|\{\mathbf{x}_i\}) &= \operatorname{argmax}_{\alpha, \mathbf{x}} C \lim_{A \to \infty} \exp -\beta E_{eff}(\alpha, \mathbf{x}, \{U^*\}, \{V^*\}) \\
&= (\operatorname{argmax}_\alpha \lim_{A \to \infty} \exp -\beta E_{eff}(\alpha, \mathbf{x}^*, \{U^*\}, \{V^*\}), \mathbf{x}^*)
\end{aligned} \tag{19}$$

where the saddle points satisfy the neural net fixed point equations

$$\begin{aligned}
(\partial E_{eff}/\partial U = 0 :) \quad V_{m,i} &= \exp U_{m,i} / \sum_j \exp U_{m,j} \\
(\partial E_{eff}/\partial V = 0 :) \quad U_{m,i} &= -\beta \left[\frac{1}{2N\sigma_r^2} |\mathbf{x}|^2 + \frac{1}{2\sigma_{jt}^2} |\mathbf{x}_i - \mathbf{x} - \mathbf{u}_m^\alpha|^2 + A(\sum_n V_{n,i} - 1) \right] \\
(\partial E_{eff}/\partial \mathbf{x} = 0 :) \quad \mathbf{x} &= \frac{1}{N(1 + (\sigma_{jt}/\sigma_r)^2)} \sum_{mi} V_{mi} (\mathbf{x}_i - \mathbf{u}_m^\alpha).
\end{aligned} \tag{20}$$

Convergent descent dynamics for such networks may be found in (Hopfield, 1984; Peterson and Soderberg, 1989) and many others. The maximization with respect to α can be handled by making one copy of this neural net for each model and adding a winner-take-all circuit.

The method introduces an asymmetry between m and i indices by imposing the $\sum_i P_{m,i} = 1$ constraint exactly but imposing the $\sum_m P_{m,i} = 1$ constraint only in the limit of infinite A. This asymmetry may be removed by *changing variables* before beginning the above calculation: let $P_{m,i} = \sum_a R_{m,a} S_{a,i}$ where R and S are two permutation matrices. Summing over P configurations is equivalent to summing over R and S configurations, for there is a 1-to-$N!$ correspondance. Then impose the $\sum_m$ and $\sum_i$ constraints exactly, as above, or else impose both $\sum_a$ constraints exactly. Either scheme preserves i-m symmetry. To finally reduce the products of continuous versions of the discrete R and S variables to linear form, one may use the objective function transformations of (Mjolsness and Garrett, 1990), e.g.

$$\sum_{mai} \hat{R}_{ma} \hat{S}_{ai} \mathbf{x}_i \cdot \mathbf{u}_m \to \sum_{ma} \hat{R}_{ma} \mathbf{u}_m \cdot (\sigma_a - \tau_a) + \sum_{ai} \hat{S}_{ai} \mathbf{x}_i \cdot (\sigma_a - \omega_a) + \text{ symmetric potential terms.} \tag{21}$$

The result is a symmetric neural net architecture for the same problem, posessing the same type of Mean Field Theory derivation from a grammar as the does the previous, asymmetric network for approximately summing over P configurations.

2.5 Approximate Neural Network without Match Variables

Short of approximating a P configuration sum via Mean Field Theory and neural nets (Section 2.4 above), there is a simpler, cheaper, less accurate approximation that we have used on matching problems

similar to the model recognition problem (find α and $\mathbf{x}$) for the dot-matching grammar. From equations (11) and (13),

$$\begin{aligned}
\Pr'(\alpha,\mathbf{x}|\{\mathbf{x}_i\}) &= C \sum_{\left\{\begin{smallmatrix}P|\ P\ \text{is a}\\ \text{permutation}\end{smallmatrix}\right\}} \exp -\sum_{mi} P_{mi}\left(\frac{1}{2N\sigma_r^2}|\mathbf{x}|^2+\frac{1}{2\sigma_{jt}^2}|\mathbf{x}_i-\mathbf{x}-\mathbf{u}_m^\alpha|^2\right)\\
&\approx C \sum_{\left\{\begin{smallmatrix}P|P_{mj}\in\{0,1,\ldots N\}\\ \wedge\sum_j P_{mj}=N\end{smallmatrix}\right\}} \exp -\sum_{mi} P_{mi}\left(\frac{1}{2N\sigma_r^2}|\mathbf{x}|^2+\frac{1}{2\sigma_{jt}^2}|\mathbf{x}_i-\mathbf{x}-\mathbf{u}_m^\alpha|^2\right)\\
&\approx \frac{C}{N!}\sum_{\{P|\sum P=N\}}\binom{N}{P_{11}\ldots P_{NN}}\prod_{mi}\left[\exp-\left(\frac{1}{2N\sigma_r^2}|\mathbf{x}|^2+\frac{1}{2\sigma_{jt}^2}|\mathbf{x}_i-\mathbf{x}-\mathbf{u}_m^\alpha|^2\right)\right]^{P_{mi}}\\
&\qquad (\textit{since almost all the multinomials are} = N!)\\
&= C'\left[\sum_{mi}\exp-\left(\tfrac{1}{2N\sigma_r^2}|\mathbf{x}|^2+\tfrac{1}{2\sigma_{jt}^2}|\mathbf{x}_i-\mathbf{x}-\mathbf{u}_m^\alpha|^2\right)\right]^N
\end{aligned} \tag{22}$$

The key step is the approximation of the sum over permutation matrices with a sum over a superset, namely all $N \times N$ nonnegative-integer-valued matrices whose entries sum to N. Among such matrices, the vast majority have low occupancy for most rows and columns. This is an entropy argument in favor of the approximation. There is also an energy argument: multiple assignments are allowed but discouraged by the effective energy term $(1/2\sigma_{jt}^2)\sum_{mi}|\mathbf{x}_i-\mathbf{x}-\mathbf{u}_m^\alpha|^2$ unless two values of $\mathbf{x}_i$ or two values of $\mathbf{u}_m$ happen to be within σ_{jt} of each other. Finally notice that the insertion of the multinomial factor improves this approximation rather than further degrading it, since configurations with $P_{mi} > 1$, not present in the original sum over permutation matrices, are weighted less strongly than those in which every P element is 0 or 1.

Under this approximation,

$$\text{argmax}_{\alpha,\mathbf{x}}\Pr(\alpha,\mathbf{x}|\{\mathbf{x}_i\}) \approx \text{argmax}_{\alpha,\mathbf{x}}\sum_{mi}\exp-\left(\frac{1}{2N\sigma_r^2}|\mathbf{x}|^2+\frac{1}{2\sigma_{jt}^2}|\mathbf{x}_i-\mathbf{x}-\mathbf{u}_m^\alpha|^2\right). \tag{23}$$

This objective function has a simple interpretation when $\sigma_r \to \infty$: it minimizes the Euclidean distance between two Gaussian-blurred images containing the $\mathbf{x}_i$ dots and a shifted version of the $\mathbf{u}_m$ dots

respectively:

$$\begin{aligned}
&\text{argmin}_{\alpha,\mathbf{x}}\int dz\,|G*I_1(z)-G*I_2(z-\mathbf{x})|^2\\
=\ &\text{argmin}_{\alpha,\mathbf{x}}\int dz\left|G_{\sigma/\sqrt{2}}*\textstyle\sum_i\delta(z-\mathbf{x}_i)-G_{\sigma/\sqrt{2}}*\sum_m\delta(z-\mathbf{x}-\mathbf{u}_m^\alpha)\right|^2\\
=\ &\text{argmin}_{\alpha,\mathbf{x}}\int dz\left|\textstyle\sum_i\exp\left(-\frac{1}{\sigma^2}|z-\mathbf{x}_i|^2\right)-\sum_m\exp\left(-\frac{1}{\sigma^2}|z-\mathbf{x}-\mathbf{u}_m^\alpha|^2\right)\right|^2\\
=\ &\text{argmin}_{\alpha,\mathbf{x}}\left[C_1-2\textstyle\sum_{mi}\int dz\exp-\frac{1}{\sigma^2}\left[|z-\mathbf{x}_i|^2+|z-\mathbf{x}-\mathbf{u}_m^\alpha|^2\right]\right]\\
=\ &\text{argmax}_{\alpha,\mathbf{x}}\textstyle\sum_{mi}\int dz\exp-\frac{1}{\sigma^2}\left[|z-\mathbf{x}_i|^2+|z-\mathbf{x}-\mathbf{u}_m^\alpha|^2\right]\\
=\ &\text{argmax}_{\alpha,\mathbf{x}}\textstyle\sum_{mi}\exp-\frac{1}{2\sigma^2}|\mathbf{x}_i-\mathbf{x}-\mathbf{u}_m^\alpha|^2
\end{aligned} \tag{24}$$

Furthermore, note that multiplying the objective in (23) by a temperature factor $\beta = 1/T$ simply rescales σ_{jt}. From this fact we conclude that deterministic annealing from $T = \infty$ down to $T = 1$, which is a good strategy for finding global maxima in equation (23), corresponds to a coarse-to-fine correlation matching algorithm: the global shift $\mathbf{x}$ is computed by repeated local optimization while gradually decreasing the Gaussian blurr parameter σ down to σ_{jt}. The output of a coarse-scale optimization is taken as the input to the next finer-scale optimization, as in deterministic annealing and other continuation methods. The resulting coarse-to-fine correlation matching algorithm is similar to the scale-space matching procedure of (Witkin et al., 1987).

The approximation (22) has the effect of eliminating the discrete P_{mi} variables, rather than replacing them with continuous versions V_{mi}. The same can be said for the "elastic net" method (Durbin and Willshaw, 1987), which is a less aggressive and probably more accurate approximation in which the sum over all permutation matrices is extended to a sum over all 0/1 matrices with exactly one nonzero element in each row but any number of nonzero entries in each column (Simic, 1990b; Yuille, 1990). Again the sum can be performed exactly. The elastic net's set of allowed matrices are intermediate in generality between permutation matrices (required in the original problem) and the far larger set of nonnegative integer matrices whose elements sum to N, used in our method. Compared to the elastic net, the present objective function is simpler, more symmetric between rows and columns, has a nicer interpretation in terms of known algorithms (correlation in scale space), and is expected to be much less accurate.

A neural net for performing the maximization of (23) with respect to $\mathbf{x}$ has been reported in (Mjolsness and Garrett, 1990). (As mentioned in Section 2.4, the maximization with respect to α can be handled by making one copy of this neural net for each model and adding a winner-take-all circuit.) σ_r

was infinite. The equations of motion were

$$\begin{aligned} \dot{\mathbf{x}} &= (1/\sigma^2)\textstyle\sum_{im} r_{im}(\mathbf{x}_i - \mathbf{u}_m - \mathbf{x}) \\ \dot{\omega}_{im} &= -\omega_{im} - (1/2\sigma^2)\textstyle\sum_{im} |\mathbf{x}_i - \mathbf{u}_m - \mathbf{x}|^2 \\ r_{im} &= \exp\omega_{im}. \end{aligned} \tag{25}$$

A continuation from large σ down to σ_{fi} was used, and it greatly reduced the network's susceptibility to finding incorrect local minima of the objective fuction.

Similar networks can be derived if the grammar includes other distortions such as dot deletion or a global rotation (see Section 4).

The networks of both this and the previous section superficially have space complexity (cost) proportional to NM where $i \in \{1 \ldots N\}$ and $m \in \{1 \ldots M\}$. It may be possible to reduce one of these cost factors, perhaps to a logarithmic term, by using more complicated architectures and approximations.

3 EXPERIMENTS IN IMAGE REGISTRATION

To demonstrate the robustness of the objective function derived in the last section, we tested it outside its formal domain of validity. This kind of robustness is often required of vision algorithms, since their (possibly implicit) mathematical models of the visual problem domain are hardly ever complete. Our grammatical models are intended to make it easier to model heterogeneous noise sources in a complex domain, but robustness would allow the grammars to remain small by modelling just the most important visual phenomena.

Equation (23) is an objective function for recovering the global two-dimensional (2D) translation of a model consisting of arbitrarily placed dots, to match up with similar dots with jittered positions. We use it instead to find the best 2D *rotation* and horizontal translation, for two images which actually differ by a horizontal *3D* translation with roughly constant camera orientation. The images consist of *line segments* rather than single dots, some of which are *missing or extra* data. In addition, there are strong *boundary effects* due to parts of the scene being translated outside the camera's field of view. The jitter is replaced by whatever positional inaccuracies come from an actual camera producing an 128 × 128 image (VISIONS Group, 1988) which is then processed by a high quality line-segment finding algorithm (Burns, 1986). Better results would be expected of objective functions derived from grammars which explicitly model more of these noise processes, such as the grammars studied in Section 4.

Since the data to be matched are not dots but line segments, one could alter the grammar and rederive the various objective functions including (23). However one could also consider a line segment to be a dense set of dots (admittedly *not* jittered randomly) and replace the sum over dot pairs in (23) with a *sum over line segment* pairs, each of which is an integral over dot pairs. For line-line or line-dot matches the integrals can be done exactly (lines are unbounded in both directions). For segment-segment (or segment-dot) matches the integrals can be approximated. First note that

$$\Theta(t) \equiv \begin{cases} 1 & \text{if } t \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

$$\Theta(t)\Theta(1-t) \approx \sum_{i=1}^{3} A_i \exp -\frac{1}{2}\frac{(c_i - t)^2}{\sigma_i^2} \tag{27}$$

where by numerical minimization of the Euclidean distance between these two functions of t, the parameters may be chosen as $A_1 = A_3 = 0.800673$, $A_2 = 1.09862$, $\sigma_1 = \sigma_3 = 0.0929032$, $\sigma_2 = 0.237033$, $c_1 = 1 - c_3 = 0.116807$, and $c_2 = 0.5$. Using this approximation, the objective function becomes

double integral along the two line segments:

$$\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \exp -\frac{1}{2}\frac{(\mathbf{q}+t\mathbf{r}-u\mathbf{s})^2}{w^2}\,\Theta(t)\Theta(1-t)\Theta(u)\Theta(1-u)\,dt\,du$$

$$\approx \sum_{i,j} A_i A_j \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \exp -\frac{1}{2}\frac{(c_i-t)^2}{\sigma_i^2} \exp -\frac{1}{2}\frac{(c_j-u)^2}{\sigma_j^2} \exp -\frac{1}{2}\frac{(\mathbf{q}+t\mathbf{r}-u\mathbf{s})^2}{w^2}\,dt\,du. \tag{28}$$

Each of these nine Gaussian integrals can be done exactly. Defining

$$\mathbf{v}_{ij} = \mathbf{q} + c_i\mathbf{r} - c_j\mathbf{s} \tag{29}$$

we have a term of the form

$$\sum_{i,j=1}^{3} A_i A_j \frac{2\pi w^2\sigma_i\sigma_j}{\sqrt{(w^2+\mathbf{r}^2\sigma_i^2)(w^2+\mathbf{s}^2\sigma_j^2)-\sigma_i^2\sigma_j^2(\mathbf{r}\cdot\mathbf{s})^2}} \exp -\frac{1}{2}\frac{\mathbf{v}_{ij}^2 w^2+(\mathbf{v}_{ij}\times\mathbf{r})^2\sigma_i^2+(\mathbf{v}_{ij}\times\mathbf{s})^2\sigma_j^2}{(w^2+\mathbf{r}^2\sigma_i^2)(w^2+\mathbf{s}^2\sigma_j^2)-\sigma_i^2\sigma_j^2(\mathbf{r}\cdot\mathbf{s})^2} \tag{30}$$

added to the objective for each pair of line segments, as was calculated by Charles Garrett (Garrett, 1990).

We experimented with minimizing this objective function with respect to unknown global translations and (sometimes) rotations, using the continuation method and sets of line segments derived from real images. The optimization method used at each scale σ was (a) for recovering translation alone, the Conjugate Gradient method with line search; and (b) for recovering translation and rotation, coordinate relaxation by repeated line searches along the x, y, and θ directions, cyclically. The results are shown in Figures 3, 4 and 5.

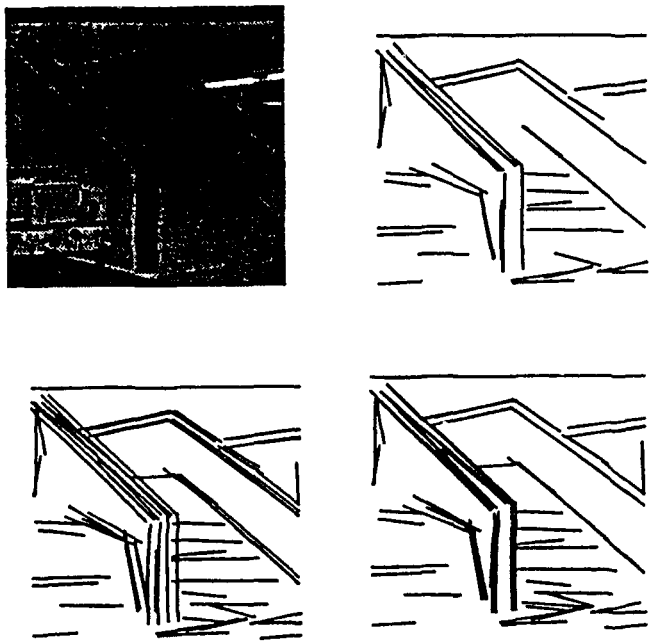


Figure 3: A simple image registration problem. (a) Stair image.(b) Long line segments derived from stair image. (c) Two unregistered line segment images derived from two images taken from two horizontally translated viewpoints in three dimensions. The images are a pair of successive frames in an image sequence. (d) Registered viersions of same data: superposed long line segments extracted from two stair images (taken from viewpoints differing by a small horizontal translation in three dimensions) that have been optimally registered in two dimensions.

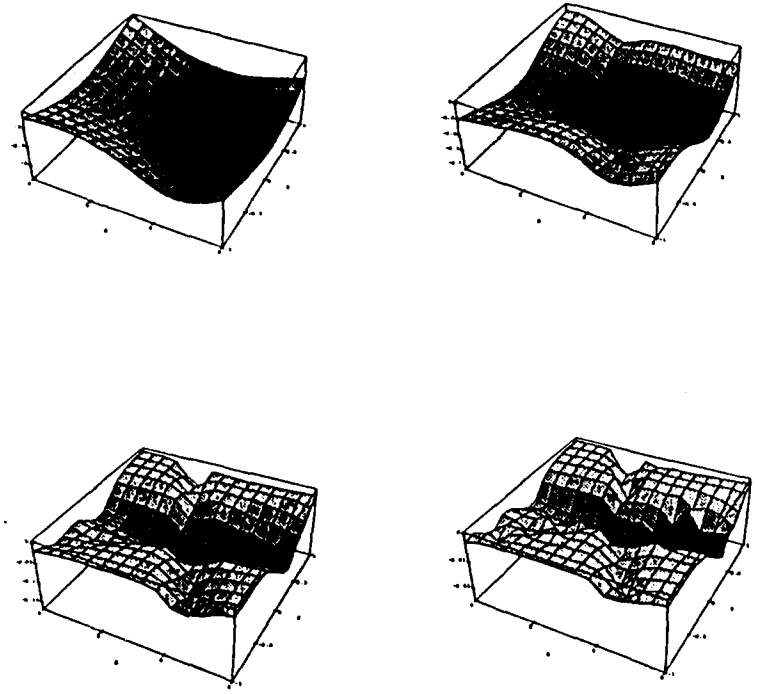


Figure 4: Continuation method (deterministic annealing). (a) Objective function at $\sigma = .0863$. (b) Objective function at $\sigma = .300$. (c) Objective function at $\sigma = .105$. (d) Objective function at $\sigma = .0364$.

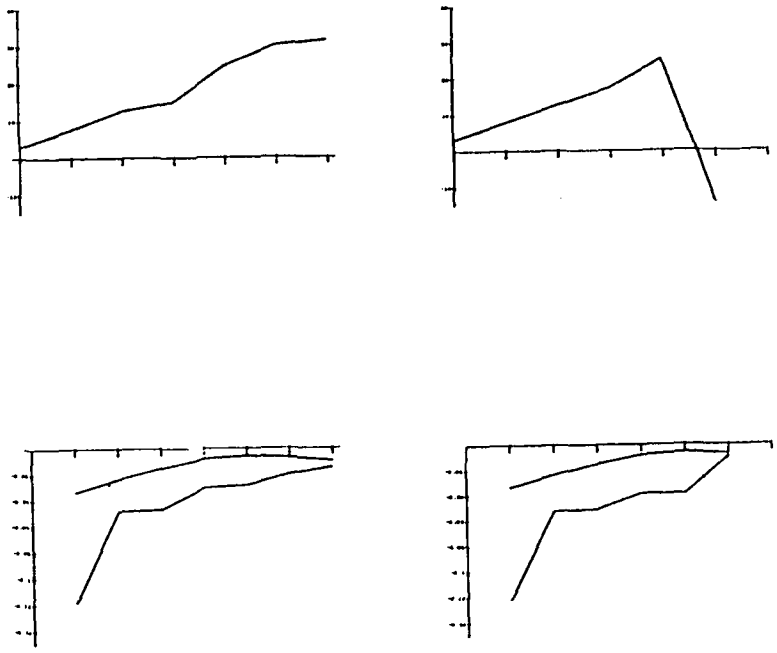


Figure 5: Image sequence displacement recovery. Frame 2 is matched to frames 3-8 in the stair imag sequence. Horizontal displacements are recovered. Other starting frames yield similar results excep for frame 1, which was much worse. (a) Horizontal displacement recovered, assuming no 2-d rotation Recovered dispacement as a function of frame number is monotonic. (b) Horizontal displacement recov ered, along with 2-d rotation which is found to be small except for the final frame. Displacements are i qualitative agreement with (a), more so for small displacements. (c) Objective function before and afte displacement is recovered (upper and lower curves) without rotation. Note gradual decrease in ΔE wit frame number (and hence with displacement). (d) Objective function before and after displacement r recovered (upper and lower curves) with rotation.

4 MORE GRAMMARS

To illustrate the generality of the grammatical method for posing vision problems in a form from which neural networks can be derived, we exhibit several grammars of increasing complexity. In Section 4.1 we add rotation and dot deletion as new sources of noise for the random-dot grammar considered in Section 2. Section 4.2 introduces a two-level hierarchy, in which models are sets of clusters of dots. Section 4.3 introduces a grammar for multiple curves in a single image, each of which is represented in the image as a set of dots that may be hard to group into their original curves. This grammar illustrates how flexible objects can be handled in our formalism.

We approach a modest plateau of generality with the grammar of Section 4.4 which again describes hierarchical dot patterns but adds multiple objects in a single scene. This degree of complexity is sufficient to introduce many interesting questions of knowledge representation in high-level vision, such as multiple instances of a model in a scene, as well as requiring segmentation and grouping as part of the recognition process. We prove that the grammatical approach can yield neural networks nearly identical to the "Frameville" neural networks we have previously studied as a means of mixing simple Artificial Intelligence frame systems (or semantic networks) with optimization-based neural networks. What is more, the transformation leading to Frameville is very natural. It simply pushes the permutation matrix as far back into the grammar as possible. This transformation should perhaps be done as a matter of course if one is looking for a modular decomposition of a large vision problem into smaller, more homogeneous pieces to which special methods such as scale-space correlation are most likely to apply.

4.1 A Grammar with 2-D Rotation and Dot Deletion

We can easily add two-dimensional rotations to the previous grammar (and similarly, other parameterized group distortions as well). The grammar, which also adds a dot deletion rule which changes the constraints on P_{im}, is shown below.

model locations	Γ^0 :	root $\rightarrow$	instance of model α at $\mathbf{x}$
		$E_0(\mathbf{x})$ =	$\frac{1}{2\sigma_r^2}\lvert\mathbf{x}\rvert^2$
rotated, jittered dot locations	Γ^1 :	instance$(\alpha,\mathbf{x})$ $\rightarrow$	$\{\text{predot}(\alpha,m,\mathbf{x}_m)\}$
		$E_1(\{\mathbf{x}_\epsilon\})$ =	$\frac{1}{2\sigma_{jt}^2}\sum_\epsilon \lvert\mathbf{x}_m-\mathbf{x}-R(\theta)\cdot\mathbf{u}_m^\alpha\rvert^2$, where $<\mathbf{u}_m^\alpha>_m=0$
dot deletion	Γ^2 :	predot$(\alpha,m,\mathbf{x}_m)$ $\rightarrow$	$\begin{cases}\text{dot}(m,\mathbf{x}_m) & \text{if } \omega_m=1;\\ \text{nothing} & \text{if } \omega_m=0.\end{cases}$
		$E_2(\omega_m)$ =	$\mu_m\omega_m$
scramble all dots	Γ^3 :	$\{\text{dot}(m,\mathbf{x}_m)\}$ $\rightarrow$	$\{\text{imagedot}(\mathbf{x}_i=\sum_m P_{m,i}\mathbf{x}_m)\}$
		$E_3(\{\mathbf{x}_i\})$ =	$-\log\prod_i \delta(\mathbf{x}_i-\sum_m P_{m,i}\mathbf{x}_m)$ where $\sum_i P_{m,i}=\omega_m \ \wedge\ \sum_m P_{m,i}=1$

(31)

The corresponding probability distribution is

$$\Pr^I(\alpha,\{\mathbf{x}_i\}) = \frac{1}{\lambda}\left(\frac{1}{\sqrt{2\pi}\sigma_r}\right)^2\left(\frac{1}{\sqrt{2\pi}\sigma_{jt}}\right)^{2N}\sum_{\left\{P\mid \sum_i P_{m,i}\le 1 \text{ and} \sum_m P_{m,i}=1\right\}} e^{-\sum_{m,i}P_{m,i}\left(\frac{1}{2N\sigma_r^2}\lvert\mathbf{x}\rvert^2+\frac{1}{2\sigma_{jt}^2}\lvert\mathbf{x}_i-\mathbf{x}-R(\theta)\cdot\mathbf{u}_m^\alpha\rvert^2+\mu_m\right)} \tag{32}$$

This is closely related to the objective function recommended for rigid body feature matching in (Yuille, 1990).

As in Section 2.4 one may approximate the maximization of the integrated probability $\Pr^3(\alpha,\mathbf{x},\theta\mid\{\mathbf{x}_i\})$ with respect to α, $\mathbf{x}$, and θ via a neural net objective function

$$E_{eff}(\alpha,\mathbf{x},\theta,\{U\},\{V\}) \equiv \sum_{m,i}V_{m,i}\left(\frac{1}{2N\sigma_r^2}\lvert\mathbf{x}\rvert^2+\frac{1}{2\sigma_{jt}^2}\lvert\mathbf{x}_i-\mathbf{x}-R(\theta)\cdot\mathbf{u}_m^\alpha\rvert^2\right)+(A/2)\sum_i(\sum_m V_{m,i}-1)^2 + (1/\beta)\sum_{m,i}U_{m,i}V_{m,i}-(1/\beta)\sum_m\log\left(\sum_i\exp U_{m,i}\right). \tag{33}$$

Alternatively, as in Section 2.5 one could use an objective function without match variables:

$$E_{eff}(\alpha,\mathbf{x},\theta)=\sum_{mi}\exp-\left(\frac{1}{2N\sigma_r^2}\lvert\mathbf{x}\rvert^2+\frac{1}{2\sigma_{jt}^2}\lvert\mathbf{x}_i-\mathbf{x}-R(\theta)\cdot\mathbf{u}_m^\alpha\rvert^2\right). \tag{34}$$

4.2 A Two-Level Random Dot Grammar

A fundamental capability gained by using a grammar is the ability to describe complex objects and scenes. As a first step in this direction, consider an object with a hierarchical decomposition into parts, with internal degrees of freedom describing the relative positions of the parts. For random dot features, the resulting images will generally be clusters of dots with unpredictable jitter of both the dot and the cluster positions. A careful model of such an object is given by this grammar:

model locations	Γ^0 : root	$\rightarrow$ instance of model α at $\mathbf{x}$
	$E_0(\mathbf{x})$	$= \frac{1}{2\sigma_r^2}\lvert\mathbf{x}\rvert^2$
cluster locations	Γ^1 : instance$(\alpha, \mathbf{x})$	$\rightarrow \{\text{clusterloc}(\alpha, c, \dot{\mathbf{x}}_c = \mathbf{x} + \mathbf{u}_c^\alpha)\}$
	$E_1(\{\dot{\mathbf{x}}_c\})$	$= -\log \prod_c \delta(\dot{\mathbf{x}}_c - \mathbf{x} - \mathbf{u}_c^\alpha)$, where $< \mathbf{u}_c^\alpha >_c = 0$
cluster jitter	Γ^2 : clusterloc$(\alpha, c, \dot{\mathbf{x}}_c)$	$\rightarrow$ cluster$(\alpha, c, \mathbf{x}_c)$
	$E_2(\mathbf{x}_c)$	$= \frac{1}{2\sigma_{cd}^2}\lvert\mathbf{x}_c - \dot{\mathbf{x}}_c\rvert^2$
dot locations	Γ^3 : cluster$(\alpha, c, \mathbf{x}_c)$	$\rightarrow \{\text{dotloc}(\alpha, c, m, \dot{\mathbf{x}}_{cm} = \mathbf{x}_c + \mathbf{u}_{cm}^\alpha)\}$
	$E_3(\{\dot{\mathbf{x}}_{cm}\})$	$= -\log \prod_m \delta(\dot{\mathbf{x}}_{cm} - \mathbf{x} - \mathbf{u}_{cm}^\alpha)$, where $< \mathbf{u}_{cm}^\alpha >_m = 0$
dot jitter	Γ^4 : dotloc$(\alpha, c, m, \dot{\mathbf{x}}_{cm})$	$\rightarrow$ dot$(c, m, \mathbf{x}_{cm})$
	$E_4(\dot{\mathbf{x}}_{cm})$	$= \frac{1}{2\sigma_{jt}^2}\lvert\mathbf{x}_{cm} - \dot{\mathbf{x}}_{cm}\rvert^2$
scramble all dots	Γ^5 : $\{\text{dot}(c, m, \mathbf{x}_{cm})\}$	$\rightarrow \{\text{imagedot}(\mathbf{x}_i = \sum_{cm} P_{cm,i}\mathbf{x}_{cm})\}$
	$E_5(\{\mathbf{x}_i\})$	$= -\log \Pr(P) \prod_i \delta(\mathbf{x}_i - \sum_{cm} P_{cm,i}\mathbf{x}_{cm})$
		where $\sum_{cm} P_{cm,i} = 1 \ \wedge \ \sum_{cm} P_{cm,i} = 1$

(35)

which is equivalent, by integrating out $\dot{\mathbf{x}}_c$ and $\dot{\mathbf{x}}_{cm}$, to a simpler grammar:

model locations	Γ^0 : root	$\rightarrow$ instance of model α at $\mathbf{x}$
	$E_0(\mathbf{x})$	$= \frac{1}{2\sigma_r^2}\lvert\mathbf{x}\rvert^2$
jittered cluster locations	Γ^1 : instance$(\alpha, \mathbf{x})$	$\rightarrow \{\text{cluster}(\alpha, c, \mathbf{x}_c)\}$
	$E_1(\{\mathbf{x}_c\})$	$= \frac{1}{2\sigma_{cd}^2}\sum_c \lvert\mathbf{x}_c - \mathbf{x} - \mathbf{u}_c^\alpha\rvert^2$, where $< \mathbf{u}_c^\alpha >_c = 0$
jittered dot locations	Γ^2 : cluster$(\alpha, c, \mathbf{x}_c)$	$\rightarrow \{\text{dot}(c, m, \mathbf{x}_{cm})\}$
	$E_2(\{\mathbf{x}_{cm}\})$	$= \frac{1}{2\sigma_{jt}^2}\sum_m \lvert\mathbf{x}_{cm} - \mathbf{x}_c - \mathbf{u}_{cm}^\alpha\rvert^2$, where $< \mathbf{u}_{cm}^\alpha >_m = 0$
scramble all dots	Γ^3 : $\{\text{dot}(c, m, \mathbf{x}_{cm})\}$	$\rightarrow \{\text{imagedot}(\mathbf{x}_i = \sum_{cm} P_{cm,i}\mathbf{x}_{cm})\}$
	$E_3(\{\mathbf{x}_i\})$	$= -\log \prod_i \delta(\mathbf{x}_i - \sum_{cm} P_{cm,i}\mathbf{x}_{cm})$
		where $\sum_i P_{m,i} = 1 \ \wedge \ \sum_m P_{m,i} = 1$

(36)

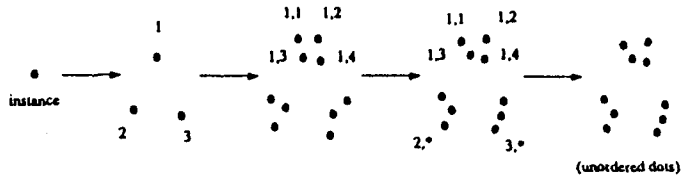


Figure 6: Operation of two-level random dot grammar. The first arrow illustrates cluster placement and cluster jitter; the next shows dot placement; the next shows dot jitter; and the final arrow is the scrambling or renumbering operation.

The corresponding probability distribution is:

$$\Pr^3(\alpha, \mathbf{x}, \{\mathbf{x}_c\}, \{\mathbf{x}_i\}) = \frac{1}{A}\left(\frac{1}{\sqrt{2\pi}\sigma_r}\right)^2\left(\frac{1}{\sqrt{2\pi}\sigma_{cd}}\right)^{2C}\left(\frac{1}{\sqrt{2\pi}\sigma_{jt}}\right)^{2N} \sum_{\left\{\begin{smallmatrix}P \mid P \text{ is a} \\ \text{permutation}\end{smallmatrix}\right\}} e^{-\sum_{cm,i} P_{cm,i}\left(\frac{1}{2N\sigma_r^2}\lvert\mathbf{x}\rvert^2 + \frac{C}{2N\sigma_{cd}^2}\lvert\mathbf{x}_c - \mathbf{x} - \mathbf{u}_c^\alpha\rvert^2 + \frac{1}{2\sigma_{jt}^2}\lvert\mathbf{x}_i - \mathbf{x}_c - \mathbf{u}_{cm}^\alpha\rvert^2\right)} \quad (37$$

where C is the number of clusters and N/C is the number of dots in each cluster. As in Section 2, one may approximate the maximization of the integrated probability $\Pr^J(\alpha, \mathbf{x}, \{\mathbf{x}_c\} \mid \{\mathbf{x}_i\})$ with respect to α, $\mathbf{x}$ and $\mathbf{x}_c$ via a neural net objective function

$$\begin{aligned} E_{eff}(\alpha, \mathbf{x}, \{\mathbf{x}_c\}, \{U\}, \{V\}) \equiv & \sum_{cm,i} V_{cm,i}\left(\frac{1}{2N\sigma_r^2}\lvert\mathbf{x}\rvert^2 + \frac{C}{2N\sigma_{cd}^2}\lvert\mathbf{x}_c - \mathbf{x} - \mathbf{u}_c^\alpha\rvert^2 + \frac{1}{2\sigma_{jt}^2}\lvert\mathbf{x}_i - \mathbf{x}_c - \mathbf{u}_{cm}^\alpha\rvert^2\right) \\ & + (A/2)\sum_i\left(\sum_{cm} V_{cm,i} - 1\right)^2 + (1/\beta)\sum_{cm,i} U_{cm,i}V_{cm,i} \\ & - (1/\beta)\sum_{cm}\log\left(\sum_i \exp U_{cm,i}\right). \end{aligned} \quad (38$$

Alternatively, as in Section 2.5 one could use an objective function without match variables:

$$E_{alt}(\alpha, \mathbf{x}, \{\mathbf{x}_c\}) = \sum_{cmi} \exp - \left(\frac{1}{2N\sigma_r^2} |\mathbf{x}|^2 + \frac{C}{2N\sigma_{cd}^2} |\mathbf{x}_c - \mathbf{x} - \mathbf{u}_c^\alpha|^2 + \frac{1}{2\sigma_{jt}^2} |\mathbf{x}_i - \mathbf{x}_c - \mathbf{u}_{cm}^\alpha|^2 \right). \quad (39)$$

Intermediate designs result from *changing variables* by separately considering the cluster and the member to which a data item is assigned: $P_{cm,i} = P_{ci}^1 P_{mi}^2$. Then P^1 could be turned into analog match variables and P^2 could be approximately ntegrated out, assuming that σ_{jt} is small with respect to the variance of model dot locations $\mathbf{u}_{cm}^\alpha$ withit a cluster, and supposing that σ_{cd} is not small with respect to the variance of cluster locations $\mathbf{u}_c^\alpha$.

4.3 Multiple Curves

This problem involves perceptual organization: one must extract multiple curves from a random-dot pattern. The grammar generates such a pattern by sequentially generating a number of curves that start with random locations and directions. Then, for each curve, the grammar sequentially generates new dot locations and curve directions according to a Markov process which favors continuity. The final picture consists of all the dots generated. This grammar is "tail-recursive", that is, a rule in the grammar can replace a term by just one term of the same type.

mak- set of curves	Γ^0 :	root	$\rightarrow$	curveset(0)
		E_0	=	0 No alternatives $\Rightarrow$ certainty.
extend curve set	Γ^1 :	curveset(c)	$\rightarrow$	$\begin{cases} \{\text{curveset}(c+1), \text{curve}(c+1, s=1, \mathbf{x}, \theta)\} & \text{if } \omega_c = 1; \\ \textit{nothing} & \text{if } \omega_c = 0. \end{cases}$
		$E_1(\mathbf{x})$	=	$\mu\omega_c + \frac{1}{2\sigma_p^2}\|\mathbf{x}\|^2$
extend curve by one dot	Γ^2 :	curve($c, s, \mathbf{x}, \theta$)	$\rightarrow$	$\begin{cases} \{\text{curve}(c, s+1, \mathbf{x}', \theta'), \text{dot}(c, s, \mathbf{x}, \theta)\} & \text{if } \omega_{cs} = 1; \\ \{\text{dot}(c, s, \mathbf{x}, \theta)\} & \text{if } \omega_{cs} = 0. \end{cases}$
		$E_2(\mathbf{x}, \theta, \mathbf{x}', \theta')$	=	$\nu\omega_{cs} + e_2(\mathbf{x} - \mathbf{x}', \theta - \theta')$, where
		$e_2(\Delta\mathbf{x}, \Delta\theta)$	$\equiv$	$\frac{1}{2\sigma_s^2}\left(\arctan\left(\frac{\Delta x_2}{\Delta x_1}\right) - \Delta\theta\right)^2 + \frac{1}{2\sigma_r^2}\left(\|\Delta\mathbf{x}\|^2 - l^2\right)^2 + \frac{1}{2\sigma_{\text{curve}}^2}(\Delta\theta)^2$
scramble all dots	Γ^3 :	$\{\text{dot}(c, s, \mathbf{x}_{cs})\}$	$\rightarrow$	$\{\text{imagedot}(\mathbf{x}_i = \sum_{cs} P_{cs,i}\mathbf{x}_{cs})\}$
		$E_3(\{\mathbf{x}_i\})$	=	$-\log \sum_{\left\{ P \mid \sum_i P_{cs,i} = A_{cs} \text{ and } \sum_{cs} P_{cs,i} = 1 \right\}} \Pr(P) \prod_i \delta(\mathbf{x}_i - \sum_{cs} P_{cs,i}\mathbf{x}_{cs})$

(40)

This grammar may be compared to the somewhat different curve grammars of (Milios, 1989).

In (Mjolsness et al., 1991a) we show that this grammar has the joint probability distribution

$$\Pr(\mathbf{x}_i, \theta_i, P_{cs,i}, C, \{S_c, c = 1, .., C\} | N) = (1 - q_1) \left[\frac{q_1(1-q_2)}{Z_1}\right]^C \left[\frac{q_2}{Z_2}\right]^N \exp(-\beta E(\{P\}, \{\mathbf{x}_i\}, \{\theta_i\})) \quad (41)$$

Table 1: Syntactical Constraints for Multiple Curves

Syntactical Constraints	Expressions
Total number of curves= C	$C = N - \sum_{i=1}^N \sum_{j=1}^N \text{next}_{ij}$
No-loop constraint	$\sum_{i=1}^N \text{next}_{ij}\text{mbr}_i = \text{mbr}_j - 1$
Start-element constraint	$\sum_{c=1}^C \text{start}_{cj} = 1 - \sum_{i=1}^N \text{next}_{ij}$
End-element constraint	$\sum_{j=1}^N \text{next}_{ij} \leq 1$
Presence-absence constraint	$\{\text{next}_{ij}\}, \{\text{start}_{ci}\} \in \{0, 1\}$ and $\{\text{mbr}_i\} \in \{1, .., N\}$

where Z_1 and Z_2 are normalization constants and

$$E(\{P\}, \{\mathbf{x}_i\}, \{\theta_i\}) = \sum_{i=1}^N \left(\sum_{c=1}^C P_{c1,i} \right) E_1(\mathbf{x}_i) + \sum_{i=1}^N \sum_{j=1}^N \left(\sum_{c=1}^C \sum_{s=1}^{S_c - 1} P_{cs,i} P_{c(s+1),j} \right) E_2(\mathbf{x}_j - \mathbf{x}_i, \theta_i, \theta_j - \theta_i) \quad (42)$$

The notation is as follows: $c = 1, .., C$ is the curve index, $s = 1, .., S_c$ is the dot index along a curve, S_c is the number of dots in curve c. The image dot locations and orientations are $\{\mathbf{x}_i\}$ and $\{\theta_i\}$. Also $q_1 = \exp(\mu)$, $q_2 = \exp(\nu)$, C is the number of curves, $P_{cs,i}$ is the permutation matrix introduced in Rule 3 of the grammar and N is the number of perceived image dots. The distribution function contains the free parameter β corresponding to the inverse of a temperature. β may by varied in a deterministic annealing process.

The energy function in equation (42) can be further simplified by a suitable change of variables. Consider the following transformations.

$$\text{next}_{ij} = \sum_{c=1}^C \sum_{s=1}^{S_c - 1} P_{cs,i} P_{c(s+1),j}, \ \text{start}_{ci} = P_{c1,i}, \ \text{mbr}_i = \sum_{c=1}^C \sum_{s=1}^{S_c} s P_{cs,i} \quad (43)$$

The choice of the new variables is not arbitrary. $\{\text{next}_{ij}\}$ tracks the membership of the data elements i and j in the same curve with the constraint that j follows i as the *next member* in the chain. $\{\text{start}_{ci}\}$ turns on, i.e. $\text{start}_{ci} = 1$, if i is the starting element of curve c. $\{\text{mbr}_i\}$ reindexes the data element i in terms of its membership number (mbr) in a curve. The membership number of the starting element in any curve c is one and for the last element in the chain, it is S_c.

The constraints needed to adequately characterize the problem undergo a transformation as well. They are given in Table 1.

In terms of the new variables, the joint probability distribution becomes

$$\Pr(\{\theta_i\},\{\text{next}_{ij}\},\{\text{start}_{\alpha i}\},\{\text{mbr}_{ij}\}|\{\mathbf{x}_i\},N)=\frac{1}{Z}\left[\exp\left(\left\{N-\sum_{i=1}^{N}\sum_{j=1}^{N}\text{next}_{ij}\right\}\log G-\beta E(\{\text{next}_{ij}\},\{\mathbf{x}_i\},\{\theta_i\})\right)\right]$$

where $G=\frac{q_1(1-p_1)}{(2\pi\sigma_\theta^2)}$ and

$$E(\{\text{next}_{ij}\},\{\mathbf{x}_i\},\{\theta_i\})=\sum_{i=1}^{N}\left(1-\sum_{j=1}^{N}\text{next}_{ji}\right)E_1(\mathbf{x}_i)+\sum_{i=1}^{N}\sum_{j=1}^{N}\text{next}_{ij}E_2(\mathbf{x}_j-\mathbf{x}_i,\theta_i,\theta_j-\theta_i). \quad (44)$$

This objective function may be transformed to a neural network as in Section 2.4, resulting in a network analogous the the Traveling Salesman network of (Hopfield and Tank, 1985) which, because of the change of variables, has the advantage that a curve can change "phase" (s-numbering as specified by {mbr}) gradually and locally as the network runs, without changing the curve's connectedness (as specified by {next}).

4.4 Frameville from a Grammar

Most neural net architectures appear inadequate for high-level vision problems because they lack the ability to express, much less use or learn, sufficiently abstract knowledge: knowledge of parameterized classes of shape, or of geometric relationships between objects, or of similarity in topology, shape or function. Just as "perceptrons" were originally intended to be minimal models of percepts, related by parameterized interconnections, one might try to invent a more abstract computational unit to model small concepts. Such a "conceptron" could only result from the combined action of many perceptrons or artificial neurons, and in this way would be a collective, large-scale phenomenon in a neural network. A conceptron would more readily map to the intuitive idea of the "concept" of an object if it: (a) could be instantiated many times in one scene or computation, with different parameters such as position and internal degrees of freedom; (b) could collect feedback from such dynamically allocated instances for use in learning; (c) could express the expected or allowed range of variation from a prototype model; (d) could enter into part-whole hierarchies with other conceptrons; (e) could enter into geometric relationships with other conceptrons; (f) could enter into generalization and specialization relationships with other conceptrons; and so forth.

The goal of the "Frameville" type of neural network architecture (Mjolsness et al., 1989; Anandan et al., 1989) is to satisfy such constraints in much the way they can be satisfied within a frame system as used in Artificial Intelligence programming (Fahlman, 1979), while exhibiting a neural substrate

or implementation which provides the kind of inexact matching abilities that objective-function based neural nets are capable of. The Frameville objective function was based on inexact graph-matching applied to a part-whole relationship denoted $INA_{\alpha\beta}\in\{0,1\}$:

$$E=\sum_{\alpha\beta}\sum_{ij}INA_{\alpha\beta}ina_{ij}M_{\alpha i}M_{\beta j}H^{\alpha\beta}(\mathbf{F}_i,\mathbf{F}_j) \quad (45)$$

subject to constraints including

$$\begin{array}{lll}\sum_\alpha INA_{\alpha\beta}M_{\alpha i} & = & \sum_j ina_{ij}M_{\beta j} \quad (a)\\ \sum_\beta INA_{\alpha\beta}M_{\beta j} & = & \sum_i ina_{ij}M_{\alpha i} \quad (b).\end{array} \quad (46)$$

(See figure 7.) Here α or β index the "frame", which could also be called the "object model", "prototype object", or "conceptron", and i or j index an instance tied to α through $M_{\alpha i}\in\{0,1\}$. $INA_{\alpha\beta}$ is assumed to be a tree in this paper (so $\sum_\alpha INA_{\alpha\beta}\le 1$) but may be a directed graph in general Frameville. $\mathbf{F}_i$ are the parameters of the instance, and $H^{\alpha\beta}$ is a distance or parameter-fit function specific to the part-whole relationship $INA_{\alpha\beta}$.

A typical use of $\mathbf{F}_i$, $\mathbf{F}_j$ and $H^{\alpha\beta}$ would be for $\mathbf{F}$ to hold environment-centered coordinates of the object i and of its part j, along with deduced object-centered coordinates such as translations and orientation angles of each part, and for $\mathbf{H}$ to perform coordinate transformations to deduce such coordinates and to check consistency between the deduced and expected object-centered coordinates of an object's parts. In this and a number of other important respects, the Frameville networks resemble the TRAFFIC system of (Zemel, 1989). Other networks are related to Frameville by virtue of the use of graph-matching or arrays of match neurons for visual object recognition (von der Malsburg and Bienenstock, 1986; von der Malsburg, 1988; Cooper, 1989; Feldman et al., 1988; Bienenstock and Doursat, 1991) or using objective functions for high-level knowledge representation (Derthick, 1988; Stolcke, 1989).

4.4.1 The Grammar

It can now be shown that the Frameville objective function and syntax constraints, as outlined above, can be derived from a random-dot grammar with multiple instances of two-level objects. We use multiple index notation $\beta\leftrightarrow(\alpha,s_2)$ (i.e. model β may occupy the s_2'th "slot" of model α, if $INA_{\alpha\beta}=1=INA_{\alpha,s_2}$) and $\gamma\leftrightarrow(\alpha,s_2,s_3)$ (i.e. model γ may occupy the s_3'th slot of model (αs_2), if $INA_{\beta\gamma}=1=INA_{\alpha s_2,s_3}$). The multiple-instance grammar is shown below.

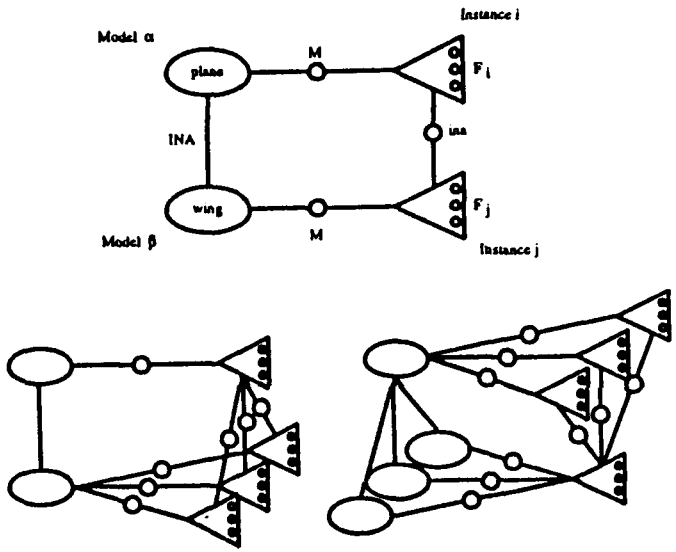


Figure 7: Frameville neural network. (a) The objective function, $E = \sum_{\alpha\beta}\sum_{ij} INA_{\alpha\beta} ina_{ij} M_{\alpha i} M_{\beta j} \times H^{\alpha\beta}(\mathbf{F}_i, \mathbf{F}_j)$. Circles are neurons, ovals are models (or frames) and triangles are model (frame) instances containing analog parameters (internal circles). (b) The constraints, $\sum_\alpha INA_{\alpha\beta} M_{\alpha i} = \sum_j ina_{ij} M_{\beta j}$ and $\sum_\beta INA_{\alpha\beta} M_{\beta j} = \sum_i ina_{ij} M_{\alpha i}$. Since $INA_{\alpha\beta}$ is a tree, the two constraint diagrams are not symmetric.

N unknown objects	Γ^0 :	root	$\rightharpoonup$	$\{\text{object}(a) \mid a \in \{1, \ldots N\}\}$
		$E_0(\mathbf{x})$	=	0
assign models and locations to objects	Γ^1 :	$\text{object}(a)$	$\rightharpoonup$	$\text{instance}(a, \alpha, \mathbf{x}_a)$
		$E_1(\alpha, \mathbf{x}_a)$	=	$\frac{1}{2\sigma_\alpha^2}\lvert\mathbf{x}_a\rvert^2$
jittered cluster locations	Γ^2 :	$\text{instance}(a, \alpha, \mathbf{x}_a)$	$\rightharpoonup$	$\{\text{cluster}(a, \alpha, s_2, \mathbf{x}_{as_2}) \mid INA_{\alpha s_2} = 1\}$
		$E_2(\{\mathbf{x}_{as_2}\})$	=	$\frac{1}{2\sigma_{s_2}^2}\sum_{s_2} INA_{\alpha s_2}\lvert\mathbf{x}_{as_2} - \mathbf{x}_a - \mathbf{u}_{s_2}^\alpha\rvert^2$
			=	$\sum_{s_2} INA_{\alpha s_2} H^{(\alpha s_2)}(\mathbf{x}_a, \mathbf{x}_{as_2})$
jittered dot locations	Γ^3 :	$\text{cluster}(a, \alpha, s_2, \mathbf{x}_{as_2})$	$\rightharpoonup$	$\{\text{predot}(a, s_2, s_3, \mathbf{x}_{as_2s_3}) \mid INA_{\alpha s_2 s_3} = 1\}$
		$E_3(\{\mathbf{x}_{as_2s_3}\})$	=	$\frac{1}{2\sigma_{s_3}^2}\sum_{s_2} INA_{\alpha s_2 s_3}\lvert\mathbf{x}_{as_2s_3} - \mathbf{x}_{as_2} - \mathbf{u}_{s_2s_3}^\alpha\rvert^2$
			=	$\sum_{s_2} INA_{\alpha s_2 s_3} H^{(\alpha s_2 s_3)}(\mathbf{x}_{as_2}, \mathbf{x}_{as_2s_3})$
dot deletion	Γ^4 :	$\text{predot}(a, \alpha, s_2, s_3, \mathbf{x}_{as_2s_3})$	$\rightharpoonup$	$\begin{cases}\text{dot}(a, \alpha, s_2, s_3, \mathbf{x}_{as_2s_3}) & \text{if } \omega_{as_2s_3} = 1; \\ \text{nothing} & \text{if } \omega_{as_2s_3} = 0.\end{cases}$
		$E_4(\omega_{as_2s_3})$	=	$INA_{\alpha s_2} INA_{as_2 s_3} \mu_{as_2s_3} \sum_a (1 - C_{a\alpha}\omega_{as_2s_3})$
scramble all dots, and add noise dots	Γ^5 :	$\{\text{dot}(a, \alpha, s_2 s_3, \mathbf{x}_{as_2s_3})\}$	$\rightharpoonup$	$\{\text{imagedot}(\mathbf{x}_k = \sum_{as_2s_3} P_{as_2s_3,k}\mathbf{x}_{as_2s_3}) \mid \omega_k = 1\}$ $\cup\{\text{imagedot}(\mathbf{x}_k) \mid \omega_k = 0\}$
		$E_5(\{\mathbf{x}_k\})$	=	$-\log\Big[\prod_{k\mid\omega_k=1} \delta(\mathbf{x}_k - \sum_{as_2s_3} P_{as_2s_3,k}\mathbf{x}_{as_2s_3})\Big]$ $-\log\delta(\sum_k \omega_k - \sum_{as_2s_3} A_{as_2s_3})$ $+\mu_{\text{extra}}\sum_k(1 - \omega_k)$
		where		$\sum_k P_{as_2s_3,k} = A_{as_2s_3}$ and $\sum_{as_2s_3} P_{as_2s_3,k} = \omega_k$

(47)

We have introduced the "aliveness variables" $A_{\cdots}^{\gamma} \in \{0, 1\}$:

$$A_{as_2s_3} = C_{a\alpha} INA_{\alpha s_2} INA_{as_2 s_3} \omega_{as_2s_3} \tag{48}$$

which is required in the expression for E_5. Here $C_{a\alpha}$ records the choice of model made in rule Γ^1 by object(a); thus $\sum_\alpha C_{a\alpha} = \Theta(a-1)\Theta(N-a)$ and $\sum_{a\alpha} C_{a\alpha} = 1$. $A_{as_2s_3}$ records which *combinations* of indices survive the whole grammar to account for some data dot.

In this grammar, Γ^0 is the essential new ingredient. Γ^4 and the "noise" dots of Γ^5 are just extra types of noise that can be handled. The following restrictions on Frameville apply for this grammar: *INA* is a tree; *ISA* is absent; sibling relationships (hence graph-matching on these relationships) are absent. Also it will turn out that the instance indices k and j are preassigned to either level 3, 2, or 1 of a hierarchy (corresponding to models indexed by α, β and γ respectively) which is not true of the original Frameville objective (45); this however is a much less substantive restriction.

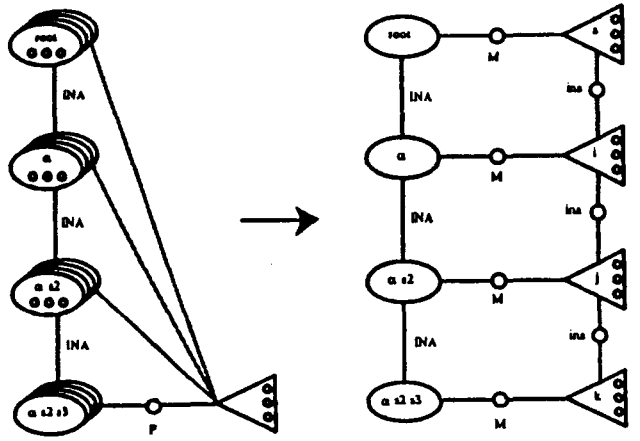


Figure 8: Change of variables, and the corresponding change in objective function, from global permutation variables P to local correspondance variables M and grouping variables $\hat{ina}$. Note that analog parameters move from the models (where they must be present in multiple copies) to the instances. Left of arrow: objective of equation (50). Right of arrow: objective of equation (45) or (92).

One useful moment of the joint model-image probability distribution is

$$\Pr^J(\{\mathbf{x}_{a\alpha}\},\{\mathbf{x}_{a\alpha s_2}\},\{C_{a\alpha}\}|\{\mathbf{x}_k\}) = \frac{1}{Z}\sum_{\left\{\begin{array}{l} P|\ \sum_k P_{a\alpha s_2 s_3,k}\le 1\ \wedge\ \sum_{a\alpha s_2 s_3} P_{a\alpha s_2 s_3,k}\le 1 \\ \wedge\ P_{a\alpha s_2 s_3,k} = P_{a\alpha s_2 s_3,k} C_{a\alpha} INA_{\alpha,s_2} INA_{\alpha s_2,s_3} \end{array}\right\}} \exp -\hat{\beta} E(\{\mathbf{x}_{a\alpha}\},\{\mathbf{x}_{a\alpha s_2}\},\{C_{a\alpha}\},\{\mathbf{x}_k\}) \tag{49}$$

where $\hat{\beta}$ is the inverse temperature (to be taken to 1) and

$$E(\ldots) = \sum_{a\alpha s_2 s_3}\sum_k P_{a\alpha s_2 s_3,k}[H^{a\alpha s_2 s_3}(\mathbf{x}_{a\alpha s_2},\mathbf{x}_k) - \mu_{\text{extra}} - \mu^{a\alpha s_2 s_3}] + \sum_{a\alpha s_2} C_{a\alpha} H^{a\alpha s_2}(\mathbf{x}_{a\alpha},\mathbf{x}_{a\alpha s_2}) + \sum_{a\alpha} C_{a\alpha} H^{root\,a}(\mathbf{x}_{a\alpha}). \tag{50}$$

This objective is illustrated in figure 8(a).

4.4.2 Changing Variables

To get the Frameville objective and constraints, we must reparameterize $\Pr^J$ and E by *changing variables*. Generally we do this by pushing the permutations, P, farther back into the grammar. A computational

advantage is that P's replacements will have fewer indices and hence be less costly. To begin with, for a data item indexed by k we could separately specify its correspondance to s_3 and to (a,α,s_2). Unfortunately E needs to know more than s_3 in order to apply the correct H term, so we instead consistently specify α, s_2, s_3 and a,α,s_2 for each k:

$$P_{a\alpha s_2 s_3,k} = M_{\alpha s_2 s_3,k}\hat{ina}_{a\alpha s_2,k} \tag{51}$$

where

$$M_{\alpha s_2 s_3,k} = \sum_a P_{a\alpha s_2 s_3,k} \quad \text{and} \quad \hat{ina}_{a\alpha s_2,k} = \sum_{s_3} P_{a\alpha s_2 s_3,k}. \tag{52}$$

The constraints on P are consistently translated into new constraints on M and $\hat{ina}$ by Lemma 1.

Lemma 1. The following two conjunctions ($a-e$ and $a'-g'$) are equivalent for 0/1 variables P, M, and $\hat{ina}$:

(a)	$M_{\alpha s_2 s_3,k} = \sum_a P_{a\alpha s_2 s_3,k}$	(a')	$P_{a\alpha s_2 s_3,k} = M_{\alpha s_2 s_3,k}\hat{ina}_{a\alpha s_2,k}$	
(b)	$\wedge\ \hat{ina}_{a\alpha s_2,k} = \sum_{s_3} P_{a\alpha s_2 s_3,k}$	(b')	$\wedge\ \sum_{s_3} M_{\alpha s_2 s_3,k} = \sum_a \hat{ina}_{a\alpha s_2,k}$	
(c)	$\wedge\ \sum_k P_{a\alpha s_2 s_3,k}\le 1$	(c')	$\wedge\ \sum_k M_{\alpha s_2 s_3,k}\hat{ina}_{a\alpha s_2,k}\le 1$	
(d)	$\wedge\ \sum_{a\alpha s_2 s_3} P_{a\alpha s_2 s_3,k}\le 1$	(d')	$\wedge\ \sum_{\alpha s_2 s_3} M_{\alpha s_2 s_3,k}\le 1$	(53)
		(e')	$\wedge\ \sum_{a\alpha s_2}\hat{ina}_{a\alpha s_2,k}\le 1$	
(e)	$\wedge\ P_{a\alpha s_2 s_3,k} = P_{a\alpha s_2 s_3,k} C_{a\alpha} INA_{\alpha,s_2} INA_{\alpha s_2,s_3}$	(f')	$\wedge\ \hat{ina}_{a\alpha s_2,k} = \hat{ina}_{a\alpha s_2,k} C_{a\alpha} INA_{\alpha,s_2}$	
		(g')	$\wedge\ M_{\alpha s_2 s_3,k} = M_{\alpha s_2 s_3,k} INA_{\alpha s_2,s_3}$	

Proof:

$\Rightarrow$:

(a'):

1. $P_{a\alpha s_2 s_3,k} = (P_{a\alpha s_2 s_3,k})^2$ (since $P\in\{0,1\}$) $\le (\sum_a P_{a\alpha s_2 s_3,k})(\sum_{s_3} P_{a\alpha s_2 s_3,k}) = M_{\alpha s_2 s_3,k}\hat{ina}_{a\alpha s_2,k}$ (by (a,b)).

2. $\sum_{s_3} P_{a\alpha s_2 s_3,k}\le\sum_{a\alpha s_2 s_3} P_{a\alpha s_2 s_3,k}\le 1$ (by (d)).

3. $\sum_{s_3} M_{\alpha s_2 s_3,k}\hat{ina}_{a\alpha s_2,k} = \sum_{s_3}\sum_{a' s_2'} P_{a'\alpha s_2 s_3,k} P_{a\alpha s_2' s_3,k}$ (by (a,b)) $= (\sum_{s_3} P_{a\alpha s_2 s_3,k})^2$ $= \sum_{s_3} P_{a\alpha s_2 s_3,k}$ (by (2)).

4. $\sum_{aa\alpha_2\alpha_3 k} |P_{aa\alpha_2\alpha_3,k} - M_{a\alpha_2\alpha_3,k} \hat{ina}_{aa\alpha_2,k}| = \sum_{aa\alpha_2\alpha_3 k} (M_{a\alpha_2\alpha_3,k} \hat{ina}_{aa\alpha_2,k} - P_{aa\alpha_2\alpha_3,k})$ (by (1)) $= 0$ (by (3)).

Thus $P_{aa\alpha_2\alpha_3,k} = M_{a\alpha_2\alpha_3,k} \hat{ina}_{aa\alpha_2,k}$.

(b') : $\sum_{\alpha_3} M_{a\alpha_2\alpha_3,k} = \sum_{\alpha\alpha_3} P_{aa\alpha_2\alpha_3,k}$ (by (a)) $= \sum_a \hat{ina}_{aa\alpha_2,k}$ (by (b)).

(c') : $\sum_k M_{a\alpha_2\alpha_3,k} \hat{ina}_{aa\alpha_2,k} = \sum_k P_{aa\alpha_2\alpha_3,k}$ (by (a')) ≤ 1 (by (c)).

(d') : $\sum_{a\alpha_2\alpha_3} M_{a\alpha_2\alpha_3,k} = \sum_{aa\alpha_2\alpha_3} P_{aa\alpha_2\alpha_3,k}$ (by (a)) ≤ 1 (by (d)).

(e') : $\sum_{aa\alpha_2} \hat{ina}_{aa\alpha_2,k} = \sum_{aa\alpha_2\alpha_3} P_{aa\alpha_2\alpha_3,k}$ (by (b)) ≤ 1 (by (d)).

(f') : $\hat{ina}_{aa\alpha_2,k} = \sum_{\alpha_3} P_{aa\alpha_2\alpha_3,k}$ (by (b)) $= C_{aa} INA_{a,\alpha_2} \sum_{\alpha_3} INA_{a\alpha_2,\alpha_3} P_{aa\alpha_2\alpha_3,k}$ (by (e)) $= (C_{aa})^2 (INA_{a,\alpha_2})^2 \sum_{\alpha_3} INA_{a\alpha_2,\alpha_3} P_{aa\alpha_2\alpha_3,k} = C_{aa} INA_{a,\alpha_2} \sum_{\alpha_3} P_{aa\alpha_2\alpha_3,k}$ (by (e)) $= C_{aa} INA_{a,\alpha_2} \hat{ina}_{aa\alpha_2,k}$ (by (b)).

(g') : $M_{a\alpha_2\alpha_3,k} = \sum_a P_{aa\alpha_2\alpha_3,k}$ (by (b)) $= INA_{a\alpha_2,\alpha_3} \sum_a P_{aa\alpha_2\alpha_3,k}$ (by (e)) $= M_{a\alpha_2\alpha_3,k} INA_{a\alpha_2,\alpha_3}$ (by (b)).

$\Leftarrow$:

(a) : $\sum_a P_{aa\alpha_2\alpha_3,k} = M_{a\alpha_2\alpha_3,k} \sum_a \hat{ina}_{aa\alpha_2,k}$ (by (a')) $= M_{a\alpha_2\alpha_3,k} (\sum_{\alpha_3'} M_{a\alpha_2\alpha_3',k})$ (by (b')) $= M_{a\alpha_2\alpha_3,k}$ (since $\sum_{\alpha_3'} M_{a\alpha_2\alpha_3',k} \leq 1$ by (d')).

(b) : $\sum_{\alpha_3} P_{aa\alpha_2\alpha_3,k} = \hat{ina}_{aa\alpha_2,k} \sum_{\alpha_3} M_{a\alpha_2\alpha_3,k}$ (by (a')) $= \hat{ina}_{aa\alpha_2,k} \sum_{a'} \hat{ina}_{a'a\alpha_2,k}$ (by (b')) $= \hat{ina}_{aa\alpha_2,k}$ (by (e')).

(c) : $\sum_k P_{aa\alpha_2\alpha_3,k} = \sum_k M_{a\alpha_2\alpha_3,k} \hat{ina}_{aa\alpha_2,k}$ (by (a')) ≤ 1 (by (c')).

(d) : $\sum_{aa\alpha_2\alpha_3} P_{aa\alpha_2\alpha_3,k} = \sum_{a\alpha_2} (\sum_{\alpha_3} M_{a\alpha_2\alpha_3,k}) (\sum_a \hat{ina}_{aa\alpha_2,k})$ (by (a')) $= \sum_{a\alpha_2} (\sum_{\alpha_3} M_{a\alpha_2\alpha_3,k})^2$ (by (b')) $= \sum_{a\alpha_2\alpha_3} M_{a\alpha_2\alpha_3,k}$ (by (d')) ≤ 1 (by (d')).

(e) : $P_{aa\alpha_2\alpha_3,k} = M_{a\alpha_2\alpha_3,k} \hat{ina}_{aa\alpha_2,k}$ (by (a')) $= M_{a\alpha_2\alpha_3,k} INA_{a\alpha_2,\alpha_3} \hat{ina}_{aa\alpha_2,k} C_{aa} INA_{a,\alpha_2}$ (by (f', g')) $= P_{aa\alpha_2\alpha_3,k} C_{aa} INA_{a,\alpha_2} INA_{a\alpha_2,\alpha_3}$ (by (a')).

QED.

This change of variables is one-to-one, so E could simply be rewritten in terms of M and $\hat{ina}$. But we can do better. $\hat{ina}$ has many similarities to P with one index removed, so one could try to change variables again to remove another index. This doesn't quite work because $\hat{ina}_{aa\alpha_2,k}$ relates coarse-scale models to fine-scale data and therefore the constraints on $\hat{ina}$ are tighter than the constraints on P. So before attempting a hierarchical induction step, we factor $\hat{ina}$ into a grouping term ina_{jk} that constructs

a data hierarchy, and a coarse-scale matching matrix $P_{aa\alpha_2 j}$:

$$\hat{ina}_{aa\alpha_2,k} = \sum_j ina_{jk} P_{aa\alpha_2 j}. \tag{54}$$

The resulting change of variables is illustrated in figure 8. When augmented by the constraints stated in Lemma 2 below, this change of variables has the effect of pushing the P matrix back one level into the grammar, leaving behind Frameville variables M and ina at the bottom level (the finest scale). This entire process can be repeated inductively, as we will see.

To state the new constraints it is necessary to account for the redundancy inherent in the assignment of j indices to groupings of data indexed by k (i.e. the freedom to permute j in ina_{jk}). To this end, we will define a fixed mapping $R_{aa\alpha_2 j}$ (consistent with C_{aa}) from possible copies of high level models (i.e. tuples (a, α, s_2)) to high level frame instances (j), then permute the j's with a matrix Q in order to get $P_{aa\alpha_2,k}$. For example R could be chosen as follows. The number of nonzero $C_{aa} INA_{a\alpha_2}$ entries is $\sum_{aa\alpha_2} C_{aa} INA_{a\alpha_2}$. Lexicographically order these, and let $R_{aa\alpha_2,j} = 1 \iff j$ indexes $(aa\alpha_2)$, for which $C_{aa} INA_{aa\alpha_2} = 1$; otherwise $R = 0$. Then

$$\begin{array}{ll} (r_1) & \sum_j R_{aa\alpha_2,j} = C_{aa} INA_{a,\alpha_2} \\ (r_2) & \sum_{aa\alpha_2} R_{aa\alpha_2,j} \leq 1 \\ (r_3) & R_{aa\alpha_2,j} = R_{aa\alpha_2,j} C_{aa} INA_{a,\alpha_2}. \end{array} \tag{55}$$

Note that (r_2) can actually be refined as follows: due to the lexicographical ordering of $(aa\alpha_2)$ by R, mapping such tuples to index j, we have

$$\begin{aligned} \sum_{aa\alpha_2} R_{aa\alpha_2,j} &= \begin{cases} 1 & \text{if } 1 \leq j \leq n^{(2)}(C, INA) \\ 0 & \text{otherwise} \end{cases} \\ &= \Theta(j-1)\Theta(n^{(2)}(C, INA) - j) \end{aligned} \tag{56}$$

where as before $\Theta(x) \equiv 1$ if $x \geq 0$ and $\Theta(x) \equiv 0$ if $x < 0$. Also $n^{(2)}(C, INA)$ is the number of allowed tuples, namely

$$n^{(2)}(C, INA) = \sum_{aa\alpha_2 j} R_{aa\alpha_2,j} = \sum_{aa\alpha_2} C_{aa} INA_{a,\alpha_2}. \tag{57}$$

For this function $R(C, INA)$ we can now prove

Lemma 2. There is a 0/1-valued function $R_{aas_2,j}(C_{aa}INA_{a,s_2})$ for which the following two conjunctions ($a-g$ and $a'-h'$) are equivalent for 0/1 variables $\hat{ina}$, Q, P, ina, and M:

(a)	$P_{aas_2,j} = \sum_{j'} R_{aas_2,j'} Q_{jj'}$,	(a')	$\hat{ina}_{aas_2,k} = \sum_j ina_{jk} P_{aas_2,j}$
(b)	$\wedge$ Q is an embedded permutation, i.e. :	(b')	$\sum_j ina_{jk} \le 1$
	(b_1) $\wedge$ $\sum_j Q_{j'j} \le 1$ (b_2) $\wedge$ $\sum_{j'} Q_{j'j} = \sum_{aas_2} R_{aas_2,j} (\le 1)$ (b_3) $\wedge$ $\sum_{j'j} Q_{j'j} = \sum_{aas_2} C_{aa} INA_{a,s_2}$		
(c)	$\wedge$ $ina_{jk} = \sum_{aas_2} \hat{ina}_{aas_2,k} P_{aas_2,j}$	(c')	$\wedge$ $ina_{jk} = ina_{jk} \sum_{aas_2} P_{aas_2,j}$
		(d')	$\wedge$ $\sum_j P_{aas_2,j} = C_{aa} INA_{as_2}$
(d)	$\wedge$ $\sum_{aas_2} \hat{ina}_{aas_2,k} \le 1$	(e')	$\wedge$ $\sum_{aas_2} P_{aas_2,j} \le 1$
(e)	$\wedge$ $\hat{ina}_{aas_2,k} = \hat{ina}_{aas_2,k} C_{aa} INA_{as_2}$	(f')	$\wedge$ $P_{aas_2,j} = P_{aas_2,j} C_{aa} INA_{as_2}$
(f)	$\wedge$ $\sum_k M_{as_1s_3,k} \hat{ina}_{aas_2,k} \le 1$	(g')	$\wedge$ $\sum_k M_{as_1s_3,k} \sum_j ina_{jk} P_{aas_2,j} \le 1$
(g)	$\wedge$ $\sum_a \hat{ina}_{aas_2,k} = \sum_{s_1} M_{as_1s_3,k}$	(h')	$\wedge$ $\sum_j ina_{jk} \sum_a P_{aas_2,j} = \sum_{s_1} M_{as_1s_3,k}$

(58)

Proof: Suppose $(a-e \iff a'-f')$. Then $(f-g)$ and $(g'-h')$ are the same pair of additional constraints expressed in terms of $\hat{ina}$ and (P, ina), respectively, using a'. So it suffices to prove $(a-e \iff a'-f')$.

$\Rightarrow$:

(d') : $\sum_j P_{aas_2,j} = \sum_{j'} R_{aas_2,j'} \sum_j Q_{jj'}$ (by (a)) $= \sum_{j'} R_{aas_2,j'} \sum_{a's_1'} R_{a'a's_1',j'}$ (by (b_2)) $= \sum_{j'} R_{aas_2,j'}$ (by (r_2)) $= C_{aa} INA_{a,s_2}$ (by (r_1)).

(e') : $\sum_{aas_2} P_{aas_2,j} = \sum_{j'} \sum_{aas_2} R_{aas_2,j'} Q_{jj'}$ (by (a)) $\le \sum_{j'} Q_{jj'}$ (by (r_2)) ≤ 1 (by (b_1)).

(a') : $\sum_j ina_{jk} P_{aas_2,j} = \sum_j \sum_{a's_1'} \hat{ina}_{a'a's_1',k} P_{a'a's_2',j'} P_{aas_2,j'}$ (by (c))

$= \sum_j \sum_{a's_1'} \hat{ina}_{a'a's_1',k} \delta_{aa'} \delta_{aa} \delta_{s_2s_2'} P_{aas_2,j}$ (by (e')) $= \hat{ina}_{aas_2,k} C_{aa} INA_{a,s_2}$ (by (d')) $= \hat{ina}_{aas_2,k}$ (by (e)).

(b') : $\sum_j ina_{jk} = \sum_{aas_2} \hat{ina}_{aas_2,k} (\sum_j P_{aas_2,j})$ (by (c)) $\le \sum_{aas_2} \hat{ina}_{aas_2,k}$ (by (d')) ≤ 1 (by (d)).

(f') : $P_{aas_2,j} = \sum_{j'} R_{aas_2,j'} Q_{jj'}$ (by (a)) $= C_{aa} INA_{a,s_2} \sum_{j'} R_{aas_2,j'} Q_{jj'}$ (by (r_3)) $= P_{aas_2,j} C_{aa} INA_{a,s_2}$ (by (a)).

(c') : $ina_{jk} \sum_{aas_2} P_{aas_2,j} = \sum_{aas_2} \sum_{a's_1'} \hat{ina}_{a'a's_1',k} P_{a'a's_2',j'} P_{aas_2,j}$ (by (c))

$= \sum_{aas_2} \sum_{a's_1'} \hat{ina}_{a'a's_1',k} \delta_{aa'} \delta_{aa} \delta_{s_2s_2'} P_{aas_2,j}$ (by (e')) $= ina_{j,k}$ (by (c)).

$\Leftarrow$: We exhibited $R(C, INA)$ obeying $(r_1)-(r_3)$. Define

$$Q_{jj'} = \sum_{aas_2} P_{aas_2,j} R_{aas_2,j'}. \tag{59}$$

(a) : $\sum_{j'} R_{aas_2,j'} Q_{jj'} = \sum_{a's_1'} P_{a'a's_1',j} \sum_{j'} R_{a'a's_1',j'} R_{aas_2,j'}$ (by (59)) $= P_{aas_2,j} \sum_{j'} R_{aas_2,j'}$ (by (r_2)) $= P_{aas_2,j} C_{aa} INA_{a,s_2}$ (by (r_1)) $= P_{aas_2,j}$ (by (f')).

(b_1) : $\sum_{j'} Q_{jj'} = \sum_{aas_2} P_{aas_2,j} \sum_{j'} R_{aas_2,j'}$ (by (59)) $= \sum_{aas_2} P_{aas_2,j} C_{aa} INA_{a,s_2}$ (by (r_1))

$= \sum_{aas_2} P_{aas_2,j}$ (by (f')) ≤ 1 (by (e')).

(b_2) : $\sum_j Q_{jj'} = \sum_{aas_2} (\sum_j P_{aas_2,j}) R_{aas_2,j'}$ (by (59)) $= \sum_{aas_2} R_{aas_2,j'} C_{aa} INA_{a,s_2}$ (by (d'))

$= \sum_{aas_2} R_{aas_2,j'}$ (by (r_3)) ≤ 1 (by (r_2)).

(b_3) : $\sum_{jj'} Q_{jj'} = \sum_{aas_2} (\sum_j P_{aas_2,j}) (\sum_{j'} R_{aas_2,j'})$ (by (59)) $= \sum_{aas_2} (C_{aa} INA_{a,s_2})^2$ (by (r_1, d'))

$= \sum_{aas_2} C_{aa} INA_{a,s_2}$.

(c) : $\sum_{aas_2} \hat{ina}_{aas_2,k} P_{aas_2,j} = \sum_{j'} ina_{j'k} \sum_{aas_2} P_{aas_2,j} P_{aas_2,j'}$ (by (a')) $= \sum_{j'} ina_{j'k} \sum_{aas_2} P_{aas_2,j} \delta_{jj'}$

(by (d')) $= ina_{jk} \sum_{aas_2} P_{aas_2,j} = ina_{jk}$ (by (d')).

(d) : $\sum_{aas_2} \hat{ina}_{aas_2,k} = \sum_j ina_{jk} \sum_{aas_2} P_{aas_2,j}$ (by (a')) $= \sum_j ina_{jk}$ (by (c')) ≤ 1 (by (b')).

(e) : $\hat{ina}_{aas_2,k} = \sum_j ina_{jk} P_{aas_2,j}$ (by (a')) $= C_{aa} INA_{a,s_2} \sum_j ina_{jk} P_{aas_2,j}$ (by (f'))

$= \hat{ina}_{aas_2,k} C_{aa} INA_{a,s_2}$.

QED

Of the constraints ($58a'-f'$), most can be regarded as constraining the variables M and ina rather than $P_{aas_2,j}$. That is not true for $d'-f'$ which, however, are analogous to the original constraints on $P_{aas_2s_3,k}$, namely ($53c-e$). Aside from dropping an index, the only change between constraints on $P_{aas_2s_3,k}$ and $P_{aas_2,j}$ is that $\sum_k P_{aas_2s_3,k} \le 1$ becomes $\sum_j P_{aas_2,j} = C_{aa} INA_{as_2}$ since the grammar generates extra (spurious) dots but not extra higher-level objects. Because of this close analogy, we can iterate the entire process once to drop the s_2 index from P, and then again to finally eliminate P and C in favor of high-level M and ina variables. What are the resulting constraints on M and ina? The answer is provided by Theorem 1.

Theorem 1. For configurations of 0/1 variables $P_{\alpha\alpha_2\alpha_3,k}$ and $C_{\alpha\alpha}$, satisfying the constraints

$$\mathcal{A}: \quad \begin{array}{rcll} \sum_k P_{\alpha\alpha_2\alpha_3,k} & \le & 1, & \\ \sum_{\alpha\alpha_2\alpha_3} P_{\alpha\alpha_2\alpha_3,k} & \le & 1, & (a) \text{ and } \sum_\alpha C_{\alpha\alpha} = 1, \ (b) \\ P_{\alpha\alpha_2\alpha_3,k} & = & P_{\alpha\alpha_2\alpha_3,k} C_{\alpha\alpha} INA_{\alpha,\alpha_2} INA_{\alpha\alpha_2,\alpha_3} & \end{array} \quad (60)$$

along with auxiliary variables Q^2 and Q^1 satisfying

$$\mathcal{B}: \quad \begin{array}{rcll} \sum_j Q^2_{j'j} & \le & 1, & (a2) \\ \sum_{j'} Q^2_{j'j} & = & \theta(n^{(2)}(C) - j)\theta(j-1), & (b2) \\ \sum_{j'j} Q^2_{j'j} & = & n^{(2)}(C) & (c2) \end{array} \quad \text{and} \quad \begin{array}{rcll} \sum_i Q^1_{i'i} & \le & 1 & (a1) \\ \sum_{i'} Q^1_{i'i} & = & \theta(n^{(1)}(C) - i)\theta(i-1) & (b1) \\ \sum_{i'i} Q^1_{i'i} & = & n^{(1)}(C) & (c1) \end{array} \quad (61)$$

(for certain integer-valued functions $n^{(1,2)}(C)$) there is a one-to-one correspondance with constrained configurations of 0/1 variables $M^3, M^2, M^1, ina^3, ina^2, ina^1$. The correspondance is given by

$$\mathcal{C}: \quad \begin{array}{rcll} P_{\alpha\alpha_2\alpha_3,k} & = & M^3_{\alpha\alpha_2\alpha_3,k} \sum_j ina^3_{jk} P_{\alpha\alpha_2,j} & (3) \\ P_{\alpha\alpha_2,j} & = & M^2_{\alpha\alpha_2,j} \sum_i ina^2_{ij} P_{\alpha,i} & (2) \\ P_{\alpha,i} & = & M^1_{\alpha,i} ina^1_{\alpha i} & (1) \end{array} \quad (62)$$

and inversely by

$$\mathcal{D}: \quad \begin{array}{rcll} M^3_{\alpha\alpha_2\alpha_3,k} & = & \sum_\alpha P_{\alpha\alpha_2\alpha_3,k} & (a3) \\ ina^3_{jk} & = & \sum_{\alpha\alpha_2} (\sum_{\alpha_3} P_{\alpha\alpha_2\alpha_3,k}) P_{\alpha\alpha_2,j} & (b3) \\ P_{\alpha\alpha_2,j} & \equiv & \sum_{j'} R^3_{\alpha\alpha_2,j}(\{C\}) Q^2_{j'j} & (c3) \end{array} \quad \begin{array}{rcll} M^2_{\alpha\alpha_2,j} & = & \sum_\alpha P_{\alpha\alpha_2,j} & (a2) \\ ina^2_{ij} & = & \sum_{\alpha\alpha} (\sum_{\alpha_2} P_{\alpha\alpha_2,j}) P_{\alpha,i} & (b2) \\ P_{\alpha,i} & \equiv & \sum_{i'} R^2_{\alpha,i}(\{C\}) Q^1_{i'i} & (c2) \end{array}$$

$$\begin{array}{rcll} M^1_{\alpha,i} & = & \sum_\alpha P_{\alpha,i} & (a1) \\ ina^1_{\alpha i} & = & \sum_\alpha P_{\alpha,i} & (b1) \end{array} \quad (63)$$

for 0/1 variables $P_{\alpha\alpha_2,j}$ and $P_{\alpha,i}$ and certain functions $R(C)$. The constraints on M and ina are

$\mathcal{E}$:

$\sum_{\alpha\alpha_2\alpha_3} M^3_{\alpha\alpha_2\alpha_3,k} \le 1$ (a3) $\sum_{\alpha\alpha_2} M^2_{\alpha\alpha_2,j} \le 1$ (a2) $\sum_\alpha M^1_{\alpha,i} \le 1$ (a1)	$\sum_j ina^3_{jk} \le 1$ (b3) $\sum_i ina^2_{ij} \le 1$ (b2) $\sum_\alpha ina^1_{\alpha,i} \le 1$ (b1)
$\sum_j ina^3_{jk} M^2_{\alpha\alpha_2,j} = \sum_{\alpha_3} INA_{\alpha\alpha_2,\alpha_3} M^3_{\alpha\alpha_2\alpha_3,k}$ (c3) $\sum_i ina^2_{ij} M^1_{\alpha,i} = \sum_{\alpha_2} INA_{\alpha,\alpha_2} M^2_{\alpha\alpha_2,j}$ (c2) $\sum_\alpha ina^1_{\alpha,i} M^0_{root,\alpha} = \sum_\alpha INA_{root,\alpha} M^1_{\alpha,i}$ (c1)	$\sum_k ina^3_{jk} M^3_{\alpha\alpha_2\alpha_3,k} \le INA_{\alpha\alpha_2,\alpha_3} M^2_{\alpha\alpha_2,j}$ (d3) $\sum_j ina^2_{ij} M^2_{\alpha\alpha_2,j} = INA_{\alpha,\alpha_2} M^1_{\alpha,i}$ (d2) $\sum_i ina^1_{\alpha,i} M^1_{\alpha,i} = C_{\alpha\alpha} \le 1$ (d1)
$ina^3_{jk} \le \sum_{\alpha\alpha_2} M^2_{\alpha\alpha_2,j}$ (e3) $ina^2_{ij} \le \sum_\alpha M^1_{\alpha,i}$ (e2) $\sum_i ina^1_{\alpha,i} = 1$ (e1)	$M^3_{\alpha\alpha_2\alpha_3,k} = INA_{\alpha\alpha_2,\alpha_3} M^3_{\alpha\alpha_2\alpha_3,k}$ (f3) $M^2_{\alpha\alpha_2,j} = INA_{\alpha,\alpha_2} M^2_{\alpha\alpha_2,j}$ (f2)

(64)

Notes. (1) The proof is given at a very detailed level, and could perhaps be shortened considerably by the invention of new notation. (2) In M, ina space, $\mathcal{E}(d1)$ has the effect of defining C in the change of variables back to P, C space. (3) The theorem asserts the existence of functions $n^{(1,2)}(C)$ and $R(C)$. We have already specified $n^{(2)} = \sum_{\alpha\alpha_2} C_{\alpha\alpha} INA_{\alpha,\alpha_2}$ and R; they also depend on the constants INA_{α,α_2}. $n^{(1)}$ is analogous to $n^{(2)}$: $n^{(1)} = \sum_{\alpha\alpha} C_{\alpha\alpha} \times (INA_{root,\alpha} = 1)$. (4) The constants $INA_{root,\alpha} = 1$ and $M^0_{root,\alpha} = 1$ (in $\mathcal{E}$) are introduced for uniformity of notation across levels 1, 2, and 3. In particular $INA_{root,\alpha} = 1$ for the *possible* models types that may appear in a scene, but $INA_{\alpha,\alpha_2} = 1$ and $INA_{\alpha\alpha_2,\alpha_3} = 1$ for the *expected* subparts of an object.

Proof of Theorem 1: The goal is to prove $\mathcal{A} \wedge \mathcal{B} \wedge \mathcal{D} \iff \mathcal{C} \wedge \mathcal{E}$, in which each labelled proposition (equations (60)-(64) above) is itself a conjunction. It will be helpful to use the phrase "... in the context of ..." technically: "$\mathcal{X} \iff \mathcal{Y}$ in the context of $\mathcal{Z}$" means $\mathcal{X} \wedge \mathcal{Z} \iff \mathcal{Y} \wedge \mathcal{Z}$.

Step 1. Assumptions $\mathcal{A}(a) \wedge \mathcal{B}(2) \wedge \mathcal{D}(3)$, along with the definition (53b), are equivalent by Lemmas 1 and 2 to (53a') $\wedge$ (53d') $\wedge$ (53g') $\wedge$ (58a' − h'). Note that $\mathcal{B}(b2)$ has been refined from (b2) in Lemma 2, according to equation (56), which is valid since Lemma 2 was proved for a particular $R(C, INA)$ satisfying equation (56). Upon eliminating ina by (58a'), this conjunction may be rewritten as the conjunction of

$$\mathcal{F}: \quad \begin{array}{rcll} P_{\alpha\alpha_2\alpha_3,k} & = & M^3_{\alpha\alpha_2\alpha_3,k} \sum_j ina^3_{jk} P_{\alpha\alpha_2,j} & (\equiv \mathcal{C}(3) \text{ from (53a')}) \\ \sum_{\alpha\alpha_2\alpha_3} M^3_{\alpha\alpha_2\alpha_3,k} & \le & 1 & (\equiv \mathcal{E}(a3) \text{ from (53d')}) \\ M^3_{\alpha\alpha_2\alpha_3,k} & = & INA_{\alpha\alpha_2,\alpha_3} M^3_{\alpha\alpha_2\alpha_3,k} & (\equiv \mathcal{E}(f3) \text{ from (53g')}) \\ \sum_j ina^3_{jk} & \le & 1 & (\equiv \mathcal{E}(b3) \text{ from (58b')}) \\ \sum_j ina^3_{jk} \sum_\alpha P_{\alpha\alpha_2,j} & = & \sum_{\alpha_3} M^3_{\alpha\alpha_2\alpha_3,k} & (\iff \mathcal{E}(c3) \text{ in context of } \mathcal{D}(a2); \text{ from (58h')}) \end{array} \quad (65)$$

and

$$\mathcal{G}: \quad \begin{array}{rcll} ina^3_{jk} & = & ina^3_{jk} \sum_{\alpha\alpha_2} P_{\alpha\alpha_2,j} & (a) \text{ (from 58(c'))} \\ \sum_k M^3_{\alpha\alpha_2\alpha_3,k} \sum_j ina^3_{jk} P_{\alpha\alpha_2,j} & \le & 1 & (b) \text{ (from 58(g'))} \end{array} \quad (66)$$

and

$$\mathcal{H}: \quad \begin{array}{rcll} \sum_j P_{\alpha\alpha_2,j} & = & C_{\alpha\alpha} INA_{\alpha,\alpha_2} & \text{(from 58(d'))} \\ \sum_{\alpha\alpha_2} P_{\alpha\alpha_2,j} & \le & 1 & \text{(from 58(e'))} \\ P_{\alpha\alpha_2,j} & = & P_{\alpha\alpha_2,j} C_{\alpha\alpha} INA_{\alpha,\alpha_2} & \text{(from 58(f'))} \end{array} \quad (67)$$

Thus $\mathcal{A}(a) \wedge \mathcal{B}(2) \wedge \mathcal{D}(3)$ is equivalent to $\mathcal{F} \wedge \mathcal{G} \wedge \mathcal{H}$.

Step 2. $\mathcal{F}$ has already been "translated" into parts of $\mathcal{C} \wedge \mathcal{E}$. We must now translate $\mathcal{G}$ and $\mathcal{H}$. Let us first work on $\mathcal{H}$, which is analogous to $\mathcal{A}(a)$ except for the modification to $\sum_j P$. There are correspouding versions of Lemmas 1 and 2, modified only by loss of an index and as follows from $\sum_j P_{a a s_2 j} = C_{aa} INA_{a,s_2}$: (53c), (53c') $\equiv$ (58f), and (58g') are all equal to $C_{aa} INA_{a,s_2}$ rather than bounded by unity. Note that for INA_{a,s_2} factors the loss of the s_2 index results in $INA_{\text{root},a} = 1$ which can be omitted from products in which it occurs. This affects (53e, f') and (58e, d', f').

Thus $\mathcal{H} \wedge \mathcal{B}(1) \wedge \mathcal{D}(2)$, along with the definition $\hat{ina}_{aa,j} = \sum_{s_2} P_{aas_2 j}$, are equivalent by analogs of Lemmas 1 and 2 to the conjunction of

$$\mathcal{I}: \quad \begin{array}{rcll} P_{aas_2 j} & = & M^2_{as_2 j} \sum_i ina^2_{ij} P_{aa,i} & (\equiv \mathcal{C}(2)) \\ \sum_{as_2} M^2_{as_2 j} & \leq & 1 & (\equiv \mathcal{E}(a2)) \\ M^2_{as_2 j} & = & INA_{a,s_2} M^2_{as_2 j} & (\equiv \mathcal{E}(f2)) \\ \sum_i ina^2_{ij} & \leq & 1 & (\equiv \mathcal{E}(b2)) \\ \sum_j ina^2_{ij} \sum_a P_{aa,i} & = & \sum_{s_2} M^2_{as_2 j} & (\iff \mathcal{E}(c2) \text{ in context of } \mathcal{D}(a1)) \end{array} \tag{68}$$

and

$$\mathcal{J}: \quad \begin{array}{rcll} ina^2_{ij} & = & ina^2_{ij} \sum_{aa} P_{aa,i} & (a) \\ \sum_j M^2_{as_2 j} \sum_i ina^2_{ij} P_{aa,i} & = & C_{aa} INA_{a,s_2} & (b) \end{array} \tag{69}$$

and

$$\mathcal{K}: \quad \begin{array}{rcll} \sum_i P_{aa,i} & = & C_{a\cdot} & (a) \\ \sum_{aa} P_{aa,i} & \leq & 1 & (b) \\ P_{aa,i} & = & P_{aa,i} C_{aa} & (c) \end{array} \tag{70}$$

Note that $P_{aa,i} = P_{aa,i} C_{aa}$ is actually redundant since $P_{aa,i} C_{aa} = P_{aa,i} \sum_{i'} P_{aa,i'} = P_{aa,i}$.

Thus $\mathcal{H} \wedge \mathcal{B}(1) \wedge \mathcal{D}(2)$ is equivalent to $\mathcal{I} \wedge \mathcal{J} \wedge \mathcal{K}$.

Step 3. At this point we know enough to rewrite $\mathcal{G}$ in terms of $\mathcal{I}$ and $\mathcal{E}$. In the context of $\mathcal{F} \wedge \mathcal{I} \wedge \mathcal{J} \wedge \mathcal{K}$, i.e. (by step 2) in the context of $\mathcal{F} \wedge \mathcal{H} \wedge \mathcal{B}(1) \wedge \mathcal{D}(2)$, $\mathcal{G}$ is equivalent to the conjunction of

$$\mathcal{L}: \quad \begin{array}{rcll} ina^3_{jk} & \leq & \sum_{as_2} M^2_{as_2 j} & (a) \quad (\equiv \mathcal{E}(c3)) \\ \sum_k M^3_{as_2 s_2,k} ina^3_{jk} & \leq & M^2_{as_2 j} INA_{as_2,s_2} & (b) \quad (\equiv \mathcal{E}(d3)) \end{array} \tag{71}$$

as we now prove.

$\mathcal{G}(a) \Rightarrow \mathcal{L}(a)$: $ina^3_{jk} = ina^3_{jk} \sum_{aas_2} P_{aas_2 j} = ina^3_{jk} \sum_{as_2} M^2_{as_2,j}$ (by $\mathcal{D}(a2)$) $\leq \sum_{as_2} M^2_{as_2,j}$ since $ina^3_{jk} \leq 1$.

$\mathcal{L}(a) \Rightarrow \mathcal{G}(a)$: $ina^3_{jk} \leq \sum_{as_2} M^2_{as_2,j} \Rightarrow (ina^3_{jk})^2 \leq ina^3_{jk} \sum_{as_2} M^2_{as_2,j} \Rightarrow ina^3_{jk} \leq ina^3_{jk} \sum_{as_2} M^2_{as_2,j} \leq ina^3_{jk}$ (since $ina^3_{jk} = (ina^3_{jk})^2$ and $\sum_{as_2} M^2_{as_2,j} \leq 1$ by $\mathcal{I}$) $\Rightarrow ina^3_{jk} = ina^3_{jk} \sum_{as_2} M^2_{as_2,j}$.

$\mathcal{L}(b) \Rightarrow \mathcal{G}(b)$: $\sum_k M^3_{as_2 s_2,k} ina^3_{jk} \leq M^2_{as_2 j} INA_{as_2,s_2} \Rightarrow \sum_k M^3_{as_2 s_2,k} \sum_j ina^3_{jk} P_{aas_2 j}$ $\leq \sum_j M^2_{as_2,j} P_{aas_2 j} INA_{as_2,s_2} = \sum_j M^2_{as_2 j} \sum_i ina^2_{ij} P_{aa,i} INA_{as_2,s_2}$ (by $\mathcal{I}$) $= C_{aa} INA_{a,s_2} INA_{as_2,s_2}$ (by $\mathcal{J}(b)$) ≤ 1.

$\mathcal{G} \Rightarrow \mathcal{L}(b)$: It suffices to deduce

$$\sum_k M^3_{as_2 s_2,k} ina^3_{jk} M^2_{as_2 j} \leq M^2_{as_2,j} INA_{as_2,s_2} \tag{72}$$

because this is $\mathcal{L}(b)$ if $M^2_{as_2 j} = 1$ and otherwise $M^2_{as_2 j} = 0 \Rightarrow P_{aas_2 j} = 0$ (by $\mathcal{I}(a)$) $\Rightarrow ina^3_{jk} = 0$ (by $\mathcal{D}(3)$, which follows from $\mathcal{F} \wedge \mathcal{G} \wedge \mathcal{H}$ and step 1) $\Rightarrow \sum_k M^3_{as_2 s_2,k} ina^3_{jk} = 0 \Rightarrow \mathcal{L}(b)$. To derive (72), multiply $\mathcal{G}(b)$ by $\sum_a P_{aas_2 j} = M^2_{as_2 j}$ (using $\mathcal{D}(2)$) and use $\mathcal{F}(c)$ to introduce INA:

$$\forall a', \quad \sum_k M^3_{as_2 s_2,k} \sum_{j'} ina^3_{j'k} P_{a'as_2 j'} \sum_a P_{aas_2 j} \leq M^2_{as_2,j} INA_{as_2,s_2}. \tag{73}$$

Now $\mathcal{H} \Rightarrow \sum_a P_{aas_2 j} \in \{0, 1\}$. Case 1: There is a unique a for which $P_{aas_2 j} = 1$. Let a' be that a. Then

$$P_{a'as_2 j'} \sum_a P_{aas_2 j} = \sum_a P_{aas_2 j'} P_{aas_2 j}. \tag{74}$$

Case 0: $P_{aas_2 j} = 0 \; \forall a \Rightarrow$ (74) for any a'. Either way the inequality (73) becomes

$$\sum_k M^3_{as_2 s_2,k} \sum_{j'} ina^3_{j'k} \sum_a P_{aas_2 j'} P_{aas_2 j} \leq M^2_{as_2,j} INA_{as_2,s_2}. \tag{75}$$

But $\mathcal{H} \Rightarrow \sum_j P_{aas_2 j} \leq 1 \Rightarrow P_{aas_2 j'} P_{aas_2 j} = \delta_{j'j} P_{aas_2 j} \Rightarrow \sum_a P_{aas_2 j'} P_{aas_2 j} = \delta_{j'j} M^2_{as_2 j}$ (by $\mathcal{D}(2)$) and (75) implies (72), as desired.

So, in the context of $\mathcal{F} \wedge \mathcal{I} \wedge \mathcal{J} \wedge \mathcal{K}$, $\mathcal{G}$ is equivalent to $\mathcal{L}$. We calculate $\mathcal{A}(a) \wedge \mathcal{B}(1,2) \wedge \mathcal{D}(2,3)$ $\equiv [\mathcal{A}(a) \wedge \mathcal{B}(2) \wedge \mathcal{D}(3)] \wedge \mathcal{B}(1) \wedge \mathcal{D}(2) \iff \mathcal{F} \wedge \mathcal{G} \wedge \mathcal{H} \wedge \mathcal{B}(1) \wedge \mathcal{D}(2)$ (by Step 1) $\iff \mathcal{D}(a2) \wedge \mathcal{F} \wedge \mathcal{G} \wedge \mathcal{H} \wedge \mathcal{B}(1) \wedge \mathcal{D}(2) \iff \mathcal{D}(a2) \wedge \mathcal{F} \wedge \mathcal{G} \wedge \mathcal{I} \wedge \mathcal{J} \wedge \mathcal{K}$ (by Step 2) $\iff \mathcal{D}(a2) \wedge \mathcal{F} \wedge \mathcal{I} \wedge \mathcal{J} \wedge \mathcal{K} \wedge \mathcal{L}$ (by Step 3 so far) $\iff \mathcal{C}(3) \wedge \mathcal{E}(3) \wedge \mathcal{I} \wedge \mathcal{J} \wedge \mathcal{K} \wedge \mathcal{D}(a2)$ (by definitions of $\mathcal{F}$ and $\mathcal{L}$, plus use of context $\mathcal{D}(a2)$ for $\mathcal{E}(c3)$.)

So, $\mathcal{A}(a) \wedge \mathcal{B}(1,2) \wedge \mathcal{D}(2,3) \iff \mathcal{C}(3) \wedge \mathcal{E}(3) \wedge \mathcal{I} \wedge \mathcal{J} \wedge \mathcal{K} \wedge \mathcal{D}(a2)$

Step 4. $\mathcal{I}$ has already been translated into the notation of $\mathcal{C} \wedge \mathcal{E}$; it remains to translate $\mathcal{J}$ and $\mathcal{K}$. In this step we work on $\mathcal{K}$. The analog of Lemma 2 breaks down here because we are at the top of the

hierarchy. Instead, observe that $\mathcal{K}(b) \wedge \mathcal{D}(1)$ is directly equivalent to the conjunction

$$\mathcal{M}: \begin{array}{rcll} P_{a\alpha,i} & = & M^1_{\alpha,i} ina^1_{ai} & (a) \quad (\equiv \mathcal{C}(1)) \\ \sum_\alpha M^1_{\alpha,i} & = & \sum_a ina^1_{ai} \le 1 & (b) \quad (\equiv \mathcal{E}(1a,b,c)), \end{array} \tag{76}$$

as we now prove.

$\Rightarrow$: $M^1_{\alpha,i} ina^1_{ai} = (\sum_b P_{b\alpha,i})(\sum_\beta P_{a\beta,i}) \ge (P_{a\alpha,i})^2 = P_{a\alpha,i}$. On the other hand $1 \ge \sum_{b\beta} P_{b\beta,i}$ (by $\mathcal{K}(b)$) $\ge \sum_b P_{b\alpha,i} + \sum_\beta P_{a\beta,i} - P_{a\alpha,i} \Rightarrow P_{a\alpha,i} + 1 \ge \sum_b P_{b\alpha,i} + \sum_\beta P_{a\beta,i} \Rightarrow (P_{a\alpha,i}+1)^2 \ge (\sum_b P_{b\alpha,i} + \sum_\beta P_{a\beta,i})^2 - (\sum_b P_{b\alpha,i} - \sum_\beta P_{a\beta,i})^2 = 4(\sum_b P_{b\alpha,i})(\sum_\beta P_{a\beta,i})$; therefore $M^1_{\alpha,i} ina^1_{ai} = (\sum_b P_{b\alpha,i})(\sum_\beta P_{a\beta,i}) = |(\sum_b P_{b\alpha,i})(\sum_\beta P_{a\beta,i})| \le |(P_{a\alpha,i}+1)^2/4| = P_{a\alpha,i}$ (since $P_{a\alpha,i} \in \{0,1\}$). Thus $M^1_{\alpha,i} ina^1_{ai} = P_{a\alpha,i}$, i.e. we have $\mathcal{M}(a)$. We see $\sum_\alpha M^1_{\alpha i} = \sum_{a\alpha} P_{a\alpha,i} = \sum_a ina^1_{ai}$ directly from $\mathcal{D}(1)$, and $\sum_{a\alpha} P_{a\alpha,i} \le 1$ is $\mathcal{K}(b)$; this establishes $\mathcal{M}(b)$.

$\Leftarrow$: $\sum_a P_{a\alpha,i} = M^1_{\alpha,i} \sum_a ina^1_{ai}$ (by $\mathcal{M}(a)$) $= M^1_{\alpha,i} \sum_\beta M^1_{\beta,i}$ (by $\mathcal{M}(b)$) $= M^1_{\alpha,i}$ (by $\mathcal{M}(b)$). Likewise $\sum_\alpha P_{a\alpha,i} = ina^1_{ai}$, so we have $\mathcal{D}(1)$. From this we calculate $\sum_{a\alpha} P_{a\alpha,i} = \sum_\alpha M^1_{\alpha,i} \le 1$ (by $\mathcal{M}(b)$).

So $\mathcal{K}(b) \wedge \mathcal{D}(1)$ is equivalent to $\mathcal{M}$. In the context of $\mathcal{M}$, $\mathcal{K}(a)$ is equivalent (by straightforward translation) to

$$\mathcal{N}: \sum_i M^1_{\alpha,i} ina^1_{ai} = C_{a\alpha} \quad (\equiv \mathcal{E}(d1)) \tag{77}$$

whence, in the context of $\mathcal{M}$, $\mathcal{A}(b)$ is equivalent to

$$\mathcal{O}: \begin{array}{rcl} 1 & = & \sum_i (\sum_\alpha M^1_{\alpha,i}) ina^1_{ai} \\ & = & \sum_i (\sum_{i'} ina^1_{ai'}) ina^1_{ai} \approx \sum_i ina^1_{ai} \quad (\equiv \mathcal{E}(e1)), \end{array} \tag{78}$$

As previously noted, $\mathcal{K}(c)$ is redundant in the context of $\mathcal{K}(a)$ and $\mathcal{K}(b)$.

Therefore, $\mathcal{A}(b) \wedge \mathcal{D}(1) \wedge \mathcal{K}$ is equivalent to $\mathcal{M} \wedge \mathcal{N} \wedge \mathcal{O}$, i.e. to $\mathcal{C}(1) \wedge \mathcal{E}(1)$.

Step 5. Here we translate $\mathcal{J}$ to provide the missing pieces of $\mathcal{C} \wedge \mathcal{E}$. In the context of $\mathcal{C}(1) \wedge \mathcal{D}(1) \wedge \mathcal{D}(2) \wedge \mathcal{E}(1) \wedge \mathcal{I} \wedge \mathcal{K} \wedge \mathcal{L}$, $\mathcal{J}$ is equivalent to the conjunction of

$$\mathcal{P}: \begin{array}{rcll} ina^2_{ij} & \le & \sum_\alpha M^1_{\alpha,i} & (a) \quad (\equiv \mathcal{E}(e2)) \\ \sum_j M^2_{\alpha s_2,j} ina^2_{ij} & = & M^1_{\alpha,i} INA_{\alpha,s_2} & (b) \quad (\equiv \mathcal{E}(d2)). \end{array} \tag{79}$$

Furthermore, the implication $\mathcal{P} \Rightarrow \mathcal{J}$ doesn't require the entire context; it follows from $\mathcal{D}(1) \wedge \mathcal{E}(a1) \wedge \mathcal{K}$ alone. We now prove these assertions.

$\mathcal{J}(a) \Rightarrow \mathcal{P}(a)$: $ina^2_{ij} = ina^2_{ij} \sum_a (\sum_\alpha P_{a\alpha,i}) = ina^2_{ij} \sum_\alpha M^1_{\alpha,i}$ (by $\mathcal{D}(a1)$, which is part of the context) $\le \sum_\alpha M^1_{\alpha,i}$ (since $ina^2_{ij} \in \{0,1\}$).

$\mathcal{P}(a) \Rightarrow \mathcal{J}(a)$: $ina^2_{ij} \le \sum_\alpha M^1_{\alpha,i} \Rightarrow (ina^2_{ij})^2 \le ina^2_{ij} \sum_\alpha M^1_{\alpha,j} \Rightarrow ina^2_{ij} \le ina^2_{ij} \sum_\alpha M^1_{\alpha,i} \le ina^2_{ij}$ (since $ina^2_{ij} = (ina^2_{ij})^2$ and $\sum_\alpha M^1_{\alpha,i} \le 1$ by $\mathcal{E}(a1)$) $\Rightarrow ina^2_{ij} = ina^2_{ij} \sum_\alpha M^1_{\alpha,j} = ina^2_{ij} \sum_{a\alpha} P_{a\alpha,i}$ (by $\mathcal{D}(a1)$).

$\mathcal{P}(b) \Rightarrow \mathcal{J}(b)$: $\sum_j M^2_{\alpha s_2,j} ina^2_{ij} = M^1_{\alpha,i} INA_{\alpha,s_2} \Rightarrow \sum_j M^2_{\alpha s_2,j} \sum_i ina^2_{ij} P_{a\alpha,i} = (\sum_i M^1_{\alpha,i} P_{a\alpha,i}) INA_{\alpha,s_2}$ $= (\sum_i M^1_{\alpha,i} M^1_{\alpha,i} ina^1_{ai}) INA_{\alpha,s_2}$ (by $\mathcal{M}$ which follows from context $\mathcal{K}(b) \wedge \mathcal{D}(1)$ by the first half of Step 4) $= C_{a\alpha} INA_{\alpha,s_2}$ (by $\mathcal{N}$ which follows from context $\mathcal{K}(a) \wedge \mathcal{M}$ i.e. $\mathcal{K}(a) \wedge \mathcal{K}(b) \wedge \mathcal{D}(1)$).

$\mathcal{J}(b) \Rightarrow \mathcal{P}(b)$: It suffices to deduce

$$\sum_j M^2_{\alpha s_2,j} ina^2_{ij} M^1_{\alpha,i} = M^1_{\alpha,i} INA_{\alpha,s_2} \tag{80}$$

because this is $\mathcal{P}(b)$ if $M^1_{\alpha,i} = 1$ and otherwise $M^1_{\alpha,i} = 0 \Rightarrow P_{a\alpha,i} = 0$ (by $\mathcal{D}(1)$) $\Rightarrow ina^2_{ij} = 0$ (by $\mathcal{D}(2)$) $\Rightarrow \sum_j M^2_{\alpha s_2,j} ina^2_{ij} = 0 \Rightarrow \mathcal{P}(b)$. To derive (80), multiply $\mathcal{J}(b)$ by $\sum_a P_{a\alpha,i} = M^1_{\alpha,i}$ (again using $\mathcal{D}(1)$)

$$\forall a', \ \sum_j M^2_{\alpha s_2,j} \sum_{i'} ina^2_{i'j} P_{a'\alpha,i'} \sum_a P_{a\alpha,i} = M^1_{\alpha,i} C_{a'\alpha} INA_{\alpha,s_2}. \tag{81}$$

Now $\mathcal{K} \Rightarrow \sum_a P_{a\alpha,i} \in \{0,1\}$.

Case 1: There is a *unique* a for which $P_{a\alpha,i} = 1$. Let a' be that a. Then

$$P_{a'\alpha,i'} \sum_a P_{a\alpha,i} = \sum_a P_{a\alpha,i'} P_{a\alpha,i}. \tag{82}$$

and $C_{a\alpha} = \sum_{i'} P_{a'\alpha,i'}$ (by $\mathcal{K}(a)$) $= 1$ (since $P_{a\alpha,i} = 1$) implies we can drop $C_{a\alpha}$ from the right hand side of (81).

Case 0: $P_{a\alpha,i} = 0$, $\forall a$. *Case* $\sum_{a'} C_{a'\alpha} > 0$: pick a' so that $C_{a'\alpha} = 1$. Then we can drop $C_{a\alpha}$ from the right hand side of (82), and $P_{a\alpha,i} = 0 \ \forall a \Rightarrow$ (82). *Case* $\sum_{a'} C_{a'\alpha} = 0$: $M^1_{\alpha i} = \sum_{a'} P_{a'\alpha,i}$ (by $\mathcal{D}(a1)$) $\le \sum_{a'i} P_{a'\alpha,i} = \sum_{a'} C_{a'\alpha}$ (by $\mathcal{K}(a)$) $= 0$. This directly implies equation (80) since both sides are zero.

So, without loss of generality we may assume equation (82) and drop $C_{a'\alpha} = 1$ from the right hand side of equation (81). As in the proof of step 3, these suffice to prove (80): (81) becomes

$$\sum_j M^2_{\alpha s_2,j} \sum_{i'} ina^2_{i'j} \sum_a P_{a\alpha,i'} P_{a\alpha,i} = M^1_{\alpha,i} INA_{\alpha,s_2} \tag{83}$$

but $\mathcal{K} \Rightarrow \sum_i P_{a\alpha,i} \le 1 \Rightarrow P_{a\alpha,i'} P_{a\alpha i} = \delta_{i'i} P_{a\alpha,i} \Rightarrow \sum_a P_{a\alpha,i'} P_{a\alpha,i} = \delta_{i'i} M^1_{\alpha,i}$ (by $\mathcal{D}(1)$) and (83) implies (80), as desired.

So, in the context of $\mathcal{C}(1) \wedge \mathcal{D}(1) \wedge \mathcal{D}(2) \wedge \mathcal{E}(1) \wedge \mathcal{I} \wedge \mathcal{K} \wedge \mathcal{L}$, $\mathcal{J}$ is equivalent to $\mathcal{P}$ which is just $\mathcal{E}(d2, e2)$.

Step 6. Here we prove that $\mathcal{A} \wedge \mathcal{B} \wedge \mathcal{D} \Rightarrow \mathcal{C} \wedge \mathcal{E}$, which is half of Theorem 1. Step 7 will establish the converse.

First note that $\mathcal{A}\wedge\mathcal{B}\wedge\mathcal{D} \iff [\mathcal{A}(a) \wedge \mathcal{B}(1,2) \wedge \mathcal{D}(2,3)] \wedge \mathcal{A}(b)\wedge\mathcal{D}(1)\wedge\mathcal{D}(a2) \iff \mathcal{C}(3)\wedge\mathcal{E}(3)\wedge\mathcal{I}\wedge\mathcal{J} \wedge [\mathcal{K} \wedge \mathcal{A}(b) \wedge \mathcal{D}(1)] \wedge \mathcal{D}(a2)$ (by Step 3) $\iff \mathcal{C}(1,3) \wedge \mathcal{E}(1,3) \wedge \mathcal{I} \wedge \mathcal{J} \wedge \mathcal{D}(a2)$ (by Step 4). This will be useful in Step 7 as well:

$$\mathcal{A} \wedge \mathcal{B} \wedge \mathcal{D} \iff \mathcal{C}(1,3) \wedge \mathcal{E}(1,3) \wedge \mathcal{I} \wedge \mathcal{J} \wedge \mathcal{D}(a2). \tag{84}$$

Next, $\mathcal{E}(3) \Rightarrow \mathcal{L}$ (by definition) and $\mathcal{C}(1) \wedge \mathcal{E}(1) \Rightarrow \mathcal{K}$ (Step 4), so we can augment (84): $\mathcal{A} \wedge \mathcal{B} \wedge \mathcal{D}$ $\Rightarrow \mathcal{C}(1,3) \wedge \mathcal{E}(1,3) \wedge \mathcal{D}(1,2) \wedge \mathcal{I} \wedge \mathcal{K} \wedge \mathcal{L} \wedge \mathcal{J} \Rightarrow \mathcal{C}(1,3) \wedge \mathcal{E}(1,3) \wedge \mathcal{D}(1,2) \wedge \mathcal{I} \wedge \mathcal{K} \wedge \mathcal{L} \wedge \mathcal{P}$ (by Step 5) $\Rightarrow \mathcal{C}(1,3) \wedge \mathcal{E}(1,3) \wedge [\mathcal{I} \wedge \mathcal{D}(a1)] \wedge \mathcal{P} \Rightarrow \mathcal{C}(1,3) \wedge \mathcal{E}(1,3) \wedge \mathcal{C}(2) \wedge \mathcal{E}(2)$ (by definition of $\mathcal{I}, \mathcal{P}$).

Thus $\mathcal{A} \wedge \mathcal{B} \wedge \mathcal{D} \Rightarrow \mathcal{C} \wedge \mathcal{E}$.

Step 7. It remains to prove $\mathcal{C} \wedge \mathcal{E} \Rightarrow \mathcal{A} \wedge \mathcal{B} \wedge \mathcal{D}$, using the previous steps. By (84) it suffices to prove $\mathcal{C} \wedge \mathcal{E} \Rightarrow \mathcal{C}(1,3) \wedge \mathcal{E}(1,3) \wedge \mathcal{I} \wedge \mathcal{J} \wedge \mathcal{D}(a2)$, which would be implied by $\mathcal{C} \wedge \mathcal{E} \Rightarrow \mathcal{I} \wedge \mathcal{J} \wedge \mathcal{D}(a2)$. So that's what we will prove.

By Step 4, $\mathcal{C}(1)\wedge\mathcal{E}(1) \equiv \mathcal{M}\wedge\mathcal{N}\wedge\mathcal{O} \Rightarrow \mathcal{D}(1)\wedge\mathcal{K}$. From the definitions of $\mathcal{I}$ and $\mathcal{P}$, $\mathcal{C}(2)\wedge\mathcal{E}(2)\wedge\mathcal{D}(1) \Rightarrow \mathcal{I}\wedge\mathcal{P}$. Thus $\mathcal{C}\wedge\mathcal{E} \Rightarrow \mathcal{D}(1)\wedge\mathcal{E}(1)\wedge\mathcal{I}\wedge\mathcal{K}\wedge\mathcal{P}$. By Step 5, in the context $\mathcal{D}(1)\wedge\mathcal{E}(a1)\wedge\mathcal{K}$ (which has just been established), $\mathcal{P} \Rightarrow \mathcal{J}$. Thus $\mathcal{C}\wedge\mathcal{E} \Rightarrow \mathcal{I}\wedge\mathcal{J}\wedge\mathcal{D}(1)$. We needed to prove $\mathcal{C}\wedge\mathcal{E} \Rightarrow \mathcal{I}\wedge\mathcal{J}\wedge\mathcal{D}(a2)$, so it now suffices to show $\mathcal{C}\wedge\mathcal{E}\wedge\mathcal{D}(1) \Rightarrow \mathcal{D}(a2)$.

Assuming $\mathcal{C}\wedge\mathcal{E}\wedge\mathcal{D}(1)$, we calculate $\sum_a P_{aas_2 j} = M^2_{as_2,j} \sum_i ina^2_{ij} \sum_a P_{aa,i}$ (by $\mathcal{C}(2)$) $= M^2_{as_2,j} \sum_i ina^2_{ij} M^1_{a,i}$ (by $\mathcal{D}(a1)$) $= M^2_{as_2,j} \sum_{s_2'} INA_{a,s_2'} M^2_{as_2',j}$ (by $\mathcal{E}(c2)$) $= M^2_{as_2,j} \sum_{s_2'} M^2_{as_2',j}$ (by $\mathcal{E}(f2)$). But $1 \geq \sum_i ina^2_{ij}$ (by $\mathcal{E}(b2)$) $\geq \sum_i ina^2_{ij} M^1_{a,i}$ (since $M^1_{a,i} \leq 1$) $= \sum_{s_2'} M^2_{as_2',j}$ (by $\mathcal{E}(c2)$ and $\mathcal{E}(f2)$). So $\sum_{s_2'} M^2_{as_2',j} \leq 1$ which implies $M^2_{as_2,j} \sum_{s_2'} M^2_{as_2',j} = M^2_{as_2,j}$, as usual. Thus $\sum_a P_{aas_2,j} = M^2_{as_2,j}$ which is $\mathcal{D}(a2)$, as desired.

Thus $\mathcal{C}\wedge\mathcal{E} \Rightarrow \mathcal{A}\wedge\mathcal{B}\wedge\mathcal{D}$.

Together, Steps 6 and 7 show that

$$\mathcal{A}\wedge\mathcal{B}\wedge\mathcal{D} \iff \mathcal{C}\wedge\mathcal{E}. \tag{85}$$

Q.E.D.

Equation (64) is to be compared with the Frameville constraints, which include (46). Those constraints are represented at three levels of hierarchical organization by $\mathcal{E}(c,d)$. In addition, constraints $\mathcal{E}(a,b)$ are conventional, appearing for example as penalty terms in "Rule 1, 2, and 3" of (Utans et al., 1989). The static constraints of $\mathcal{E}(f)$ are usually taken to be obvious: only models present in the model base may have match variables. This leaves $\mathcal{E}(e)$ as the only constraint not clearly accounted for in previous Frameville research. Conversely previous work has relied mostly on the constraints found here, including $\mathcal{E}(c,d)$; for example (Utans et al., 1989) includes $\mathcal{E}(c,d)$ as "Rules 5 and 6", leaving only the constraint of "Rule 4" to differ from $\mathcal{E}(e)$. (Rules 7-9 of that source just encoded the graph-matching objective function.) Thus, the constraints of Theorem 1 agree with those of Frameville, with a few minor differences (namely $\mathcal{E}(e)$ and the specialization of every M and ina variable to some hierarchical level) which may be improvements on previous Frameville neural networks.

4.4.3 The Objective Function

Proposition $\mathcal{E}$ in Theorem 1 establishes the Frameville syntax constraints, including the subtle constraints of equation (46), as a consequence of the grammar. We must now derive the Frameville objective function, equation (45), from equation (50) which is the objective derived from the grammar. This involves changing variables from P to ina and M as in Theorem 1, and also from analog model variables $\mathbf{x}_{aas_2}$ and $\mathbf{x}_{aa}$, which were redundantly present in multiple copies for every model, to analog instance variables $\mathbf{x}_j$ and $\mathbf{x}_i$ which determine the original variables by

$$\mathbf{x}_{aas_2} = \sum_j P_{aas_2,j}\mathbf{x}_j \quad \text{and} \quad \mathbf{x}_{aa} = \sum_i P_{aa,i}\mathbf{x}_i. \tag{86}$$

Now translating (50) is a matter of substituting new variables for old in each term and adding an entropy term that arises from integrating out Q, i.e. the redundancy of M and ina with respect to P.

The first term of (50) is $E_3 = \sum_{aas_2s_3} \sum_k P_{aas_2s_3,k} [H^{as_2s_3}(\mathbf{x}_{aas_2}, \mathbf{x}_k) - \mu_{extra} - \mu^{as_2s_3}]$. Without loss of generality we can absorb the μ's into H, and omit them from the algebra. Substituting $P_{aas_2s_3,k} = M_{as_2s_3,k} \sum_j ina_{jk} P_{aas_2,j}$ and using (86), $E_3 = \sum_{aas_2s_3} \sum_k M_{as_2s_3,k} \sum_j ina_{jk} P_{aas_2,j}$ $\times H^{as_2s_3}(\sum_{j'} P_{aas_2,j'}\mathbf{x}_{j'}, \mathbf{x}_k)$. But $H^{as_2s_3}(\sum_{j'} P_{aas_2,j'}\mathbf{x}_{j'}, \mathbf{x}_k) = \sum_{j'} P_{aas_2,j'} H^{as_2s_3}(\mathbf{x}_{j'}, \mathbf{x}_k)$. Since $P_{aas_2,j} P_{aas_2,j'} = \delta_{j,j'} P_{aas_2,j}$, we find $E_3 = \sum_{as_2s_3} \sum_k M_{as_2s_3,k} \sum_j ina_{jk} (\sum_a P_{aas_2,j}) \times H^{as_2s_3}(\mathbf{x}_j, \mathbf{x}_k)$. We can also evaluate $\sum_a P_{aas_2,j} = M_{as_2,j} \sum_a ina_{aa,j}$ (by (53a')) $= M_{as_2,j} \sum_{s_2} M_{as_2,j}$ (by (58g)) =

$M_{\alpha s_2 J}$ (by $\mathcal{E}(a2)$). Also $M_{\alpha s_2 s_3 k} = M_{\alpha s_2 s_3 k} INA_{\alpha s_2, s_3}$ (by $\mathcal{E}(f3)$). Then

$$E_3 = \sum_{\alpha s_2 s_3} \sum_{jk} M_{\alpha s_2 s_3 k} M_{\alpha s_2 J} INA_{\alpha s_2, s_3} ina_{jk} H^{\alpha s_2 s_3}(\mathbf{x}_j, \mathbf{x}_k). \tag{87}$$

The second term of (50) is $E_2 = \sum_{\alpha a s_2} C_{\alpha a} H^{\alpha s_2}(\mathbf{x}_{\alpha a}, \mathbf{x}_{a \alpha s_2}) = \sum_{\alpha a s_2} C_{\alpha a} H^{\alpha s_2}(\sum_i P_{\alpha a, i} \mathbf{x}_i, \sum_j P_{a \alpha s_2 J} \mathbf{x}_j)$ $= \sum_{\alpha s_2} \sum_{ij} (\sum_a C_{\alpha a} P_{\alpha a, i} P_{a \alpha s_2 j}) H^{\alpha s_2}(\mathbf{x}_i, \mathbf{x}_j)$. But $\sum_a C_{\alpha a} P_{\alpha a, i} P_{a \alpha s_2 j} = \sum_a P_{\alpha a, i} P_{a \alpha s_2 J}$ (by (58f')) $= \sum_a P_{\alpha a, i} M_{\alpha s_2 J} \sum_{i'} ina_{i'j} P_{\alpha a, i'} = M_{\alpha s_2 J} \sum_{i'} ina_{i'j} \sum_a P_{\alpha a, i} P_{\alpha a, i'} = M_{\alpha s_2 J} \sum_{i'} ina_{i'j} \delta_{i,i'} \sum_a P_{\alpha a, i} =$ $M_{\alpha s_2 J} ina_{ij} M_{\alpha, i}$ (by $\mathcal{D}(a1)$). Also $M_{\alpha s_2 J} = M_{\alpha s_2 J} INA_{\alpha, s_2}$ (by $\mathcal{E}(f2)$). Thus

$$E_2 = \sum_{\alpha s_2} \sum_{ij} M_{\alpha s_2 J} M_{\alpha, i} INA_{\alpha, s_2} ina_{ij} H^{\alpha s_2}(\mathbf{x}_i, \mathbf{x}_j). \tag{88}$$

The third term of (50) is $E_1 = \sum_{a \alpha} C_{a \alpha} H^{\text{root}\,\alpha}(\mathbf{x}_{a \alpha}) = \sum_{a \alpha} C_{a \alpha} H^{\text{root}\,\alpha}(\sum_i P_{a \alpha, i} \mathbf{x}_i)$ $= \sum_{a \alpha i} C_{a \alpha} P_{a \alpha, i} H^{\text{root}\,\alpha}(\mathbf{x}_i) = \sum_{a \alpha i} P_{a \alpha, i} H^{\text{root}\,\alpha}(\mathbf{x}_i)$ (by $\mathcal{K}(c)$) $= \sum_{a \alpha i} M^1_{\alpha, i} ina^1_{ai} H^{\text{root}\,\alpha}(\mathbf{x}_i)$ (by $\mathcal{C}(1)$). Thus

$$E_1 = \sum_{\alpha} \sum_{ai} M^1_{\alpha, i} M^0_{\text{root}, a} INA_{\text{root}, \alpha} ina^1_{ai} H^{\text{root}\,\alpha}(\mathbf{x}_i) \tag{89}$$

(since $M^0_{\text{root}, a} = 1$ and $INA_{\text{root}, \alpha} = 1$), which is in the desired form. But since $\sum_a P_{a \alpha, i} = M^1_{\alpha, i}$ (by $\mathcal{D}(a1)$), this special case could be more simply written as $E_1 = \sum_{\alpha i} M^1_{\alpha, i} H^{\text{root}\,\alpha}(\mathbf{x}_i)$.

There are also entropy terms that arise from integrating out Q^1 and Q^2, i.e. the redundancy of M and ina with respect to P. Let $N^2 \geq n^2(C)$ be the maximum number of level 2 instances, i.e. the range of j, and $N^1 \geq n^1(C)$ be the same for level 1. Then the entropy term (from B) is $\log(\binom{N^2}{n^2} n^2!) + \log(\binom{N^1}{n^1} n^1!) = \log((N^2!) - \log((N^2 - n^2)!) + \log(N^1!) - \log((N^1 - n^1)!)$ i.e.

$$S = \log(N^2!) - \log((N^2 - \sum_{a \alpha s_2 i} ina^1_{ai} M^1_{ai} INA_{\alpha, s_2})!) + \log(N^1!) - \log((N^1 - \sum_{ai} ina^1_{ai} M^1_{ai})!) \tag{90}$$

There is a further entropy term associated with integrating out unused analog model variables such as $\mathbf{x}_{a \alpha s_2}$, but it may be absorbed into chemical potentials (μ terms in E) and hence into H terms.

Thus the final Frameville objective function is

$$\begin{aligned} E(M, ina, \mathbf{x}) \;=\; & \textstyle\sum_{\alpha s_2 s_3} \sum_{jk} M^3_{\alpha s_2 s_3, k} M^2_{\alpha s_2, j} INA_{\alpha s_2, s_3} ina^3_{jk} H^{\alpha s_2 s_3}(\mathbf{x}_j, \mathbf{x}_k) \\ & + \textstyle\sum_{\alpha s_2} \sum_{ij} M^2_{\alpha s_2, j} M^1_{\alpha, i} INA_{\alpha, s_2} ina^2_{ij} H^{\alpha s_2}(\mathbf{x}_i, \mathbf{x}_j) \\ & + \textstyle\sum_{\alpha} \sum_{ai} M^1_{\alpha, i} M^0_{\text{root}, \alpha} INA_{\text{root}, \alpha} ina^1_{ai} H^{\text{root}\,\alpha}(\mathbf{x}_i) \\ & + 1/\beta [\log(N^2!) - \log((N^2 - \textstyle\sum_{a \alpha s_2 i} ina^1_{ai} M^1_{ai} INA_{\alpha, s_2})!) \\ & + \log(N^1!) - \log((N^1 - \textstyle\sum_{a \alpha i} ina^1_{ai} M^1_{ai})!)] \end{aligned} \tag{91}$$

where as before $M^0_{\text{root}, a} = INA_{\text{root}, \alpha} = 1$. This is a stratified or layered version of the original Frameville objective, as can be seen by rewriting it in terms of model indices $\alpha, \beta, \ldots$ that range over all three levels, and similar modified instance indices $i, j, \ldots$, and using the fact that in this paper INA is a tree:

$$\begin{aligned} E(M, ina, x) \;=\; & \textstyle\sum_{\alpha \beta} \sum_{ij} M^{\text{level}(\alpha)}_{\alpha, i} M^{\text{level}(\beta)}_{\beta, j} INA_{\alpha \beta} ina^{\text{level}(\beta)}_{ij} H^{\alpha \beta}(\mathbf{x}_i, \mathbf{x}_j) \\ & + 1/\hat{\beta} [\log(N^2!) - \log((N^2 - \textstyle\sum_{\alpha \beta i j} ina^1_{ij} M^0_{\text{root}, i} M^1_{\alpha, j} INA_{\alpha, \beta})!) \\ & + \log(N^1!) - \log((N^1 - \textstyle\sum_{\alpha i j} ina^1_{ij} M^0_{\text{root}, i} M^1_{\alpha, i})!)]. \end{aligned} \tag{92}$$

This is to be compared with equation (45). The graph-matching terms differ just by the new level superscripts on M and ina, which preallocate instance indices i, j, k to specific levels of abstraction. Such specialization of instance function could probably be removed at the cost of further entropy terms. The entropy terms are new, and easily implementable with analog neural networks by Stirling's approximation and algebraic transformations of the resulting $X \log X$ forms (Mjolsness and Garrett, 1990).

Thus we have translated the probability distribution of the Frameville grammar, specified by the objective and the constraints, into the standard Frameville variables, recovering the standard objective function terms and constraints along with a few new ones. This may be regarded as a transformation at the level of the probability distribution, before Mean Field Theory is applied and hence before any approximations are made. It may also be possible to express this derivation as a transformation at the level of the grammar, in which the permutation operation is applied in a limited form at each stage rather than globally at the final stage of the grammar.

4.5 Frameville and High-Level Vision

As mentioned earlier, with the Frameville grammar we approach a modest plateau of generality. From the generalized assignment problem of equation (50) we have derived a network which explicitly has problems of *recognition* (find M_{ai}), *segmentation* or *grouping* (find ina_{ij}), *correspondance* between data and the expected parts of an object (find $M_{\alpha s_2 s_3, k}$), *multiple instances of a model* (find $\mathbf{x}_i$ rather than, say, $\mathbf{x}_\alpha$), at multiple *levels of abstraction* (levels 3, 2, and 1 in the hierarchical grammar). These processes arise from Bayesian inference on a constrained Boltzmann probability distribution which, we proved, is equivalent to the distribution generated by a simple grammar. The transformation to Frameville is natural: it simply pushes the permutation matrix as far back into the grammar as is possible, so that each grammar rule can be regarded as having its own renumbering processes even at abstract levels.

The resulting Frameville objective is different from the original generalized assignment objective in several important ways. Where the assignment objective is linear in its binary-valued match variables,

the Frameville objective is cubic in far fewer variables. (The linear or cubic terms are multiplied by analog parameter-check objectives $H(\mathbf{x},\mathbf{x})$ in both cases.) This increase in polynomial order may create more local minima in a smaller net. It is not clear whether this is a net gain or loss for practical optimization. On the other hand, further simplifying transformations such as the correlation method of Section 2.5, which have special conditions of applicability, are far more likely to apply to small, single-object correspondance problems (e.g. find $M_{\alpha_3 i_3,k}$ given $M_{\alpha_2 j}$) that can arise in Frameville than to the original monolithic assignment problem.

Thus the Frameville formulation suggests a modular decomposition of a large vision problem into smaller, more homogeneous pieces to which special methods are most likely to apply. The decomposition follows the lines indicated by the hierarchical and heterogeneous grammar.

One important aspect of model-based vision, and of the original Frameville networks, is still missing: the use of an indexing scheme such as a discrimination tree or graph composed of *ISA*-links to organize the set of models into a data base. Another efficient indexing scheme, not used in the Frameville networks, would be geometric hashing (Lamdan et al., 1988). Either form of indexing could possibly be added as a learned computational shortcut. Learning is briefly discussed in the next section.

5 LEARNING

Three possible methods for learning a grammar are suggested here. They all assume that learning the grammar can be expressed as tuning its parameters, as is the case for unstructured neural networks. First, the kind of grammar we have been studying could be augmented with an initial set of "meta-grammar" rules, which randomly choose the parameters of the permanent models and then generate many images by the usual grammar. The task of inferring the permanent models' parameters is just another Bayesian inference problem, stretched out over many images. Second, one could minimize the Kullback information between the probability distributions of an unknown grammar, images from which the perceiver sees, and a parameterized grammar. This algorithm would be similar to the "Boltzmann machine" for neural network learning (Hinton and Sejnowski, 1986). Finally one could look for *clusters* in model space by defining a distance "metric" D between images and mathematically projecting it back though the grammar.

The latter alternative can be explained by an analogy with the forward-going composition law for probabilities (and hence objectives)

$$e^{-\beta E_{n+1}(\mathbf{x}_{n+1})} \propto \Pr(\mathbf{x}_{n+1}) = \int d\mathbf{x}_n \Pr(\mathbf{x}_{n+1}|\mathbf{x}_n)\Pr(\mathbf{x}_n) \propto \int d\mathbf{x}_n \Pr(\mathbf{x}_{n+1}|\mathbf{x}_n) e^{-\beta E_n(\mathbf{x}_n)}, \tag{93}$$

which uses only those probabilities $\Pr(\mathbf{x}_{n+1}|\mathbf{x}_n)$ directly provided by the grammar. One could define a backward-going composition law for $\hat{D}_n(\mathbf{x}_n,\mathbf{y}_n)$ by, for example,

$$e^{-\beta \hat{D}_n(\mathbf{x}_n,\mathbf{y}_n)} \propto \int d\mathbf{x}_{n+1} \int d\mathbf{y}_{n+1} \Pr(\mathbf{x}_{n+1}|\mathbf{x}_n)\Pr(\mathbf{y}_{n+1}|\mathbf{y}_n) e^{-\beta \hat{D}_{n+1}(\mathbf{x}_{n+1},\mathbf{y}_{n+1})}. \tag{94}$$

Here the proposed distance D is related to $\hat{D}$ by $\hat{D}(\mathbf{x},\mathbf{y}) = f(D(\mathbf{x},\mathbf{y}))$ where f is any monotonic, strictly increasing function such as the identity plus a constant, $\hat{D} = D + c$, or the squaring function, $\hat{D} = D^2$.

This expression for D has the advantage that β can be varied as in deterministic annealing, and also that if $\Pr(\mathbf{x}_{n+1}|\mathbf{x}_n)$ is an invariant measure on a group G (e.g. translations, rotations or permutations with uniform probability) and D_{n+1} respects G (i.e. if $\Pr(\mathbf{x}_{n+1}|\mathbf{x}_n) = \int dg \delta(\mathbf{x}_{n+1} - g\cdot\mathbf{x}_n)$ and $D_{n+1}(g\cdot x, y) = D_{n+1}(x, g^{-1}\cdot y)$) then the large-$\beta$ limit of (94) is

$$\begin{aligned} D_n(\mathbf{x}_n,\mathbf{y}_n) &= f^{-1}(-(1/\beta)\log \max_{g_1,g_2\in G} \exp -\beta f(D_{n+1}(g_1\cdot\mathbf{x}_n, g_2\cdot\mathbf{y}_n))) \\ &= \min_{g_1,g_2\in G} D_{n+1}(g_1\cdot\mathbf{x}_n, g_2\cdot\mathbf{y}_n) \\ &= \min_{g_1,g_2\in G} D_{n+1}(g_2^{-1}g_1\cdot\mathbf{x}_n, \mathbf{y}_n) \\ &= \min_{g\in G} D_{n+1}(g\cdot\mathbf{x}_n, \mathbf{y}_n) \end{aligned} \tag{95}$$

which actually preserves all the properties of a distance metric including the triangle inequality. That is, if D_{n+1} is a distance metric then D_n is also.

Expressions similar to (94) could define distances between objects at earlier and later stages in the grammar, for example between an image and a model; in this way data can be clustered in the model parameter space. In fact if d is such an asymmetric "distance" and $\Pr(\mathbf{x}_{n+1}|\mathbf{x}_n)$ is an invariant measure on a group G respected by d, then the large-β limit of

$$e^{-\beta d_n(\mathbf{x}_n,\mathbf{y}_m)} \propto \int d\mathbf{x}_{n+1} \Pr(\mathbf{x}_{n+1}|\mathbf{x}_n) e^{-\beta d_{n+1}(\mathbf{x}_{n+1},\mathbf{y}_m)}. \tag{96}$$

is again

$$d_n(\mathbf{x}_n,\mathbf{y}_m) = \min_{g\in G} d_{n+1}(g\cdot\mathbf{x}_n,\mathbf{y}_m) \tag{97}$$

where $m > n$.

5.1 Graph Matching Objectives

For example, we can use (95) with $\hat{D} = f(D) = D^2$ to define the distance between two graphs, G and g, by adding permutation invariance to the Euclidean distance $D_1(G,g)$ between the 0/1 connection matrices $G_{\alpha\beta}$ and g_{ij}. Euclidean distance between matrices respects the adjoint representation of the group of permutations, i.e. permutations that act by permuting the nodes of a graph and hence both indices of a matrix:

$$\begin{aligned}
[D_0(G,g)]^2 &= [\min_{\text{Permutations } P} D_1(G, P\circ g)]^2 \\
&= \min_{\text{Permutations } P} \|G - P\cdot g\cdot P^T\|^2 \\
&= \min_{\text{Permutations } P} \|G\cdot P - P\cdot g\|^2 \\
&= \min_{\text{Permutations } P} \left[\|G\|^2 + \|g\|^2 - 2\text{trace}\left(G\cdot Pg^T\cdot P^T\right)\right] \\
&= \|G\|^2 + \|g\|^2 - 2\min_{\text{Permutations } P} \textstyle\sum_{\alpha\beta ij} G_{\alpha\beta} P_{\beta j} g_{ij} P_{\alpha i}
\end{aligned} \tag{98}$$

which is the standard neural network objective function for inexact graph matching (von der Malsburg and Bienenstock, 1986; Hopfield and Tank, 1986), up to an additive constant. It is also a standard cost metric for missing model links and extra data links in pure graph matching for computer vision (Ballard and Brown, 1982), restricted to the case of permutation correspondances:

$$\begin{aligned}
E(P) &= \text{missing model graph links} + \text{extra data graph links} \\
&= \textstyle\sum_{\alpha\beta} \max\left(0, G_{\alpha\beta} - \sum_{ij} g_{ij} P_{\alpha i} P_{\beta j}\right) + \sum_{\alpha\beta} \max\left(0, \sum_{ij} g_{ij} P_{\alpha i} P_{\beta j} - G_{\alpha\beta}\right) \\
&= \textstyle\sum_{\alpha\beta} \left(G_{\alpha\beta} - \sum_{ij} g_{ij} P_{\alpha i} P_{\beta j}\right)^2
\end{aligned} \tag{99}$$

since G and g are 0/1 matrices and P is a permutation matrix. This is the same expression as that minimized to produce D_0 above, though it has the interpretation of a distance (squared) between model and data rather than between two models.

These objectives for graph-matching have proven useful in neural networks devoid of learning, but they may also be useful in the approach to learning a structured neural network through abstract clustering at a high level in a visual grammar.

6 DISCUSSION

Figure 1 is a diagram of the method used in this paper to derive neural networks from visual grammars via Bayesian inference. Given a grammar of suitable form, one can calculate a joint probability distribution on images and their explanations. This distribution may be transformed, by changing variables, as we showed in the multiple curve grouping and Frameville networks. By using the Mean Field Theory approximation, a Bayesian inference problem on this distribution is transformed into an optimization problem with an algebraic objective function. This function can be further transformed, for example using the techniques of (Mjolsness and Garrett, 1990), to reduce its cost or increase its circuit-level implementability; then a neural network follows from descent dynamics.

We studied grammars that model visual phenomena such as missing and extra data, group invariances, hierarchical objects, multiple instances of an object in a scene, and flexible spline-like objects. The rudiments of a frame system for knowledge representation emerged naturally from one such grammar, by pushing the matching process from low levels to high levels in a hierarchical, multiple-instance grammar. Nevertheless the full representational capacities of such grammars were hardly used: it remains to design networks from grammars that generate trees recursively, or are context-dependent, or include discrimination or inheritance trees on the set of object models. A somewhat more general view of grammars whose rules posess connectionist models (similar to the objective functions attached to rules in this paper) is presented in (Mjolsness et al., 1991b), where such grammars are proposed for modelling the development of biological organisms. A different way to translate a class of grammars into neural network objective functions is presented in (Miller et al., 1991); it currently applies to "regular" languages and has been demonstrated in a low-level vision problem. Other directions for generalization of the parallel grammar occur in the extensive literature on L-systems (Rozenberg and Salomaa, 1980) and graph grammars (Ehrig et al., 1983).

The grammars examined in this paper do not yet produced realistic images, and one could consider adding new rules to move from the "pictures" we studied to gray-level images. This would allow the preprocessing of images to produce pictures composed of image features, necessary in Section 3, to be replaced with more neural network computation. At another extreme, the idea of relating optimization to Bayesian inference on a probabilistic grammar is in principle not restricted to vision at all, and could perhaps be adapted to other problems to which neural network optimization has been applied.

The complexity of the visual world will certainly demand many grammars of increasing size for success in computer vision by our approach. We have not yet discussed how to obtain such grammars, though Figure 1 suggests hand-design and learning should be directed at producing grammars rather

than the subsequent stages. Hand design takes a lot of human labor. Fortunately the large amount of research in computer graphics is a source of generative models for images, some of which are mathematically simple enough to be put in the form of a probabilistic grammar or are already close to that form (e.g. (Prusinkiewicz, 1990),(Smith, 1984)). In addition we speculated on possible algorithms for learning the grammars, or at least aspects of them, by using visual experience.

Acknowledgements

C. Garrett performed the computer simulations. The author benefitted from discussions with P. Anandan, Gene Gindi, Greg Hager, Chien Ping Lu, Drew McDermott, Anand Rangarajan, Joachim Utans and Alan Yuille.

References

Anandan, P., Letovsky, S., and Mjolsness, E. (1989). Connectionist variable-binding by optimization. In *11th Annual Conference of the Cognitive Science Society*. Lawrence Earlbaum Associates. University of Michigan.

Ballard, D. H. and Brown, C. M. (1982). *Computer Vision*, chapter 11, pages 360–362. Prentice Hall. Equation 11.3, Missing Cost.

Bertsekas, D. P. and Tsitsiklis, J. N. (1989). *Parallel and Distributed Computation*, chapter 5, page 333. Prentice Hall.

Bienenstock, E. and Doursat, R. (1991). Issues of representation in neural networks. In Gorea, A., editor, *Representations of Vision: Trends and Tacit Assumptions in Vision Research*. Cambridge University Press.

Burns, J. B. (1986). Extracting straight lines. *IEEE Trans. PAMI*, 8(4):425–455.

Cooper, P. R. (1989). *Parallel Object Recognition from Structure (The Tinkertoy Project)*. PhD thesis, University of Rochester Department of Computer Science. Technical Report 301.

Derthick, M. (1988). *Mundane Reasoning by Parallel Constraint Satisfaction*. PhD thesis, Carnegie Mellon University. Available as CMU-CS-88-182 from the Computer Science Department.

Durbin, R. and Willshaw, D. (1987). An analog approach to the travelling salesman problem using an elastic net method. *Nature*, 326:689–691.

Ehrig, H., Nagl, M., Rosenfeld, A., and Rozenberg, G., editors (1983). *Graph Grammars and their Application to Computer Science; Third International Workshop*, volume 291 of *Lecture Notes in Computer Science*. Springer-Verlag.

Fahlman, S. E. (1979). *NETL: A System for Representing and Using Real-World Knowledge*. MIT Press.

Feldman, J. A., Fanty, M. A., and Goddard, N. H. (1988). Computing with structured neural networks. *IEEE Computer*, page 91.

Garrett, C. (1990). Private communication.

Hinton, G. E. and Sejnowski, T. J. (1986). Learning and relearning in Boltzmann machines. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing*, chapter 7. MIT Press.

Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences USA*, vol. 81:3088–3092.

Hopfield, J. J. and Tank, D. W. (1985). 'Neural' computation of decisions in optimization problems. *Biological Cybernetics*, vol. 52:141–152.

Hopfield, J. J. and Tank, D. W. (1986). Collective computation with continuous variables. In *Disordered Systems and Biological Organization*, pages 155–170. Springer-Verlag.

Kosowsky, J. J. and Yuille, A. L. (1991). The invisible hand algorithm: Solving the assignment problem with statistical physics. Technical Report 91-1, Harvard Robotics Laboratory.

Lamdan, Y., Schwartz, J. T., and Wolfson, H. J. (1988). Object recognition by affine invariant matching. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 335–344.

Milios, E. E. (1989). Shape matching using curvature processes. *Computer Vision, Graphics, and Image Processing*, 47:203–226.

Miller, M. I., Roysam, B., Smith, K. R., and O'Sullivan, J. A. (1991). Representing and computing regular languages on massively parallel networks. *IEEE Transactions on Neural Networks*, 2(1).

Mjolsness, E. and Garrett, C. (1990). Algebraic transformations of objective functions. *Neural Networks*, 3:651–669.

Mjolsness, E., Gindi, G., and Anandan, P. (1989). Optimization in model matching and perceptual organization. *Neural Computation*, 1.

Mjolsness, E., Rangarajan, A., and Garrett, C. (1991a). A neural net for reconstruction of multiple curves with a visual grammar. Manuscript in preparation. Summary in International Joint Conference on Neural Networks Seattle, July 1991.

Mjolsness, E., Sharp, D. H., and Reinitz, J. (1991b). *A connectionist model of development*. *Journal of Theoretical Biology*. In press. Also available as Yale Computer Science technical report YALEU/DCS/796, June 1990.

Peterson, C. and Soderberg, B. (1989). A new method for mapping optimization problems onto neural networks. *International Journal of Neural Systems*, 1(3).

Prusinkiewicz, P. (1990). *The Algorithmic Beauty of Plants*. Springer-Verlag New York.

Rozenberg, G. and Salomaa, A. (1980). *The Mathematical Theory of L-systems*. Academic Press.

Simic, P. D. (1990a). Constrainted nets for graph matching and other quadratic assignment problems. Technical Report CALT-68-1672, Caltech Physics Department.

Simic, P. D. (1990b). Statistical mechanics as the underlying theory of 'elastic' and 'neural' optimization. *Network: Computation in Neural Systems*, 1(1):89–103.

Smith, A. R. (1984). Plants, fractals and formal languages. *Computer Graphics*, 18(3):1–10. Proceedings of SIGGRAPH '84.

Stolcke, A. (1989). Unification as constraint satisfaction in structured connectionist networks. *Neural Computation*, 1(4):559–567.

Tresp, V. (1991). A neural network approach for three-dimensional object recognition. In *Neural Information Processing Systems 3*. Morgan Kaufmann.

Utans, J., Gindi, G., Mjolsness, E., and Anandan, P. (1989). Neural networks for object recognition within compositional hierarchies: Initial experiments. Technical Report Report No. 8903, Center for Systems Science, Yale University Department of Electrical Engineering.

Van den Bout, D. E. and Miller, III, T. K. (1990). Graph partitioning using annealed networks. *IEEE Transactions on Neural Networks*, 1(2):192–203.

VISIONS Group, U. (1988). Staircase image sequence. Test images.

von der Malsburg, C. (1988). Pattern recognition by labeled graph matching. *Neural Networks*, 1:141–148.

von der M lsburg, C. and Bienenstock, E. (1986). Statistical coding and short-term synaptic plasticity: A scheme for knowledge representation in the brain. In *Disordered Systems and Biological Organization*, pages 247–252. Springer-Verlag.

Witkin, A., Terzopoulos, D., and Kass, M. (1987). Signal matching through scale space. *International Journal of Computer Vision*, 1:133–144.

Yuille, A. L. (1990). Generalized deformable models, statistical physics, and matching problems. *Neural Computation*, 2(1):1–24.

Zemel, R. S. (1989). Traffic: A connectionist model of object recognition. Technical Report CRG-TR-89-2, University of Toronto Connectionist Research Group.